Technology
Arts Sciences
TH Köln

Technische Hochschule Köln
Campus Gummersbach
Fakultät für Informatik und Ingenieurwissenschaften
Institut für Informatik

Master's Thesis

Leistungsmessung und Leistungsbewertung von NoSQL-Datenbanken

---

Name / Matrikelnummer
Tiziano Lombardi
11038348

Erstprüferin / Zweitprüferin
Prof. Dr. Heide Faeskorn-Woyke
Prof. Dr. Birgit Bertelsmeier

Thesis vorgelegt im Studiengang Master Informatik - Software Engineering
zur Erlangung des akademischen Grades Master of Science

Gummersbach, 30. Mai 2017

Abstract

The aim of this Master's Thesis is to give an overview of the different database types and performance analyses. This comparative literature review concentrates on a young field of research and examines non-relational NoSQL databases in particular, which have become more and more popular in the last years and which offer several advantages over relational databases. But what can particular database implementations achieve with varying data models, and which experimental arrangements are appropriate for which application requirements? In the beginning this work defines criteria for the evaluation of performance and surveys the experimental approaches of several researchers. An major focus lies on assessing and providing comparability of testing methods and results. In addition to methodological procedures, an important tool to implement performance measuring into NoSQL databases is being discussed: the YCSB Framework.

Zusammenfassung

Das Ziel der vorliegenden Master's Thesis ist es, einen Überblick der verschiedenen Datenbanktypen und Leistungsanalysen zu geben. Die vergleichende Literaturstudie beschäftigt sich mit einem jungen Forschungsfeld und betrachtet insbesondere nicht-relationale NoSQL-Datenbanken, welche in den letzten Jahren immer beliebter geworden sind und einige Vorteile gegenüber relationalen Datenbanken aufweisen. Doch was können die konkreten Datenbankimplementierungen bei unterschiedlichen Datenmodellen leisten und welcher Testaufbau bietet sich bei welchen Einsatzanforderungen an? Zu Anfang definiert diese Arbeit Kriterien zur Bewertung von Leistung und untersucht experimentelle Vorgehensweise verschiedener Forscher. Ein wichtiger Fokus liegt darauf, die Vergleichbarkeit der Messmethoden und Ergebnisse einzuschätzen und zu gewährleisten. Neben dem methodischen Vorgehen wird mit dem YCSB-Framework ein wichtiges Werkzeug besprochen, mit dem Leistungsmessungen in NoSQL-Datenbanken implementiert werden können.

Inhaltsverzeichnis

1 Einleitung

Die technologischen Entwicklungen in der Informationstechnologie (IT) sind in den letzten beiden Dekaden geprägt von ungeheuren Datenmengen, welche zu verarbeiten sind, das sogenannte Big Data und den gestiegenen Anforderungen durch Benutzeranfragen (Requests) und der dadurch entstehenden Last, die IT-Systeme zu bewältigen haben. Die Rechenkapazität stieg im selben Zeitraum exponentiell an, während die Kosten für Standardhardware sanken. In diesem Kontext etablierten sich NoSQL-Datenbanksysteme als Alternative zu traditionellen relationalen Datenbanksystemen, um den Anforderungen der horizontalen Skalierbarkeit und Hochverfügbarkeit von Anwendungen, wie soziale Medien, zu begegnen.[1]

Die Ursprünge der Leistungsmessung, der Leistungsbewertung und des resultierenden Leistungsvergleich (Benchmarking) von IT-Systemen, insbesondere der Computer und Netzwerke, gehen in die 1970er Jahre zurück.[2] Die Leistungsanalyse von NoSQL-Datenbanken ist jedoch ein sehr junges Forschungsfeld, welches erst in letzter Zeit durch den massiven Einsatz von nicht-relationalen Datenbanken im Umfeld von Big Data und für entscheidungsunterstützende Systeme (DSS) an Bedeutung gewonnen hat und die Grundlage für eine Technologieentscheidung bildet.[3]

Viele der in dieser Arbeit vorgestellten Konzepte haben ihren Ursprung im Test von Software. Letztlich sind auch Messung, Bewertung und Analyse nichts weiteres als ein Prozess zur Verifikation und Validierung von wohldefinierten Anforderungen gegenüber einem Produkt, einer Lösung oder Entscheidung. Die Auswahl und Kombination der vorgestellten Konzepte, Vorgehensmodelle und Werkzeuge erfolgte anhand von Referenzen und Beispielen in wissenschaftlichen Arbeiten sowie dem Gesichtspunkt einer Anwendbarkeit und Erweiterbarkeit dieser Methoden in einer eher experimentellen Umgebung.

1.1 Ziel der Arbeit

Die Grundlage für diese Arbeit bildet die Ausarbeitung „Benchmarking cloud serving systems with YCSB“ von Cooper et al. [Coo+10] zum YCSB-Framework, dem Werkzeug für die Umsetzung von Leistungsvergleichen bei NoSQL-Datenbanksystemen. Ausgehend hiervon wird in dieser Thesis mittels einer vergleichenden Literaturanalyse versucht, die vielen Modelle und Aspekte zur Leistungsfähigkeit von NoSQL-Datenbanksystemen im Querschnitt zu betrachten. Der Fokus liegt in der Vorstellung von diesbezüglich relevanten Konzepten aus dem Themenumfeld von NoSQL, dem Vorgehen bei einer experimentellen Durchführung einer Leistungsanalyse, den Analyseverfahren, die zur Anwendung kommen, und dem Verständnis der relevanten Metriken hierzu.

Der Inhalt dieser Arbeit lässt sich insofern abgrenzen, als nicht auf konkrete Datenbankimplementierungen eingegangen wird, sondern immer nur Datenbanktypen und das Vorgehen im Mittelpunkt stehen. Ebenso werden bestimmte Themen wie der MapReduce-Algorithmus nicht betrachtet und es wird auch keine Fallstudie durchgeführt. Vielmehr

[1] Vgl. [Ges+16]
[2] Vgl. [Bou+10]
[3] Vgl. [Flo+12], [BBF14]

werden bereits veröffentlichte Fallstudien vorgestellt. Sie dienen dabei der Erläuterung der Besonderheiten bei der Leistungsmessung und der Einführung in die Analyseverfahren zur Leistungsbewertung.

1.2 Aufbau der Thesis

Der Aufbau dieser Thesis spiegelt das deduktive Vorgehen bei ihrer Erstellung wieder. Die Reihenfolge der Kapitel und Abschnitte können als roter Faden der Argumentation betrachtet werden. Im 2. Kapitel, dem Grundlagenteil, befinden sich ein kurzer historischer Abriss und eine Einordnung. Es werden zuerst die grundsätzlichen Begrifflichkeiten, Modelle und Anforderungen definiert, die im weiteren Verlauf verwendet werden und in unterschiedlichen Betrachtungsweisen wiederkehren. Oftmals geschieht dies in Abgrenzung zum relationalen Pendant. Das 3. Kapitel beinhaltet das methodische Vorgehen zu einem Leistungsvergleich. Hieran orientiert sich das konzeptionelle Vorgehen in den darauf folgenden Kapitel. Neben der Methodik widmet sich dieses Kapitel auch den Werkzeugen, wie dem YCSB-Framework und seinen Erweiterungen. Im 4. Kapitel werden anhand der eingeführten Begriffe die Dimensionen und Aspekte der Leistungsmessung aufgezeigt. Dies mündet in einem Verzeichnis aller zur Leistungsmessung relevanten Termini. Das 5. Kapitel greift die Methoden aus dem vorherigen Kapitel auf und quantifiziert die entstehenden Daten im Zusammenhang mit einer Leistungsbewertung. Ferner findet an dieser Stelle eine kritische Auseinandersetzung statt. Es ist zu erwähnen, dass bestimmte Sachverhalte nicht trennscharf der Leistungsmessung oder Leistungsbewertung zuzuordnen sind. Die Thesis schließt mit einem Fazit in Kapitel 6 ab.

2 Grundlagen

Im folgenden Grundlagenteil sollen die in dieser Arbeit verwendeten Begriffe und Modelle für ein eindeutiges und einheitliches Verständnis vorgestellt und erörtert werden. Die Abgrenzung erfolgt anhand von Definitionen und Technologien im Kontext von bestehenden Konzepten und neuen Ansätzen.

2.1 NoSQL-Datenbanken

Nicht-relationale Datenbanken in Form von hierarchischen oder netzwerkartigen Datenbanken existieren seit der Erfindung des modernen Computers in Gestalt von Mainframes.[1] Die Beschreibung des Relationenmodells, welche die theoretische Grundlage für relationale Datenbanksysteme (RDBMS) ist, erfolgte erst später im Jahr 1970 durch Ted Codd. Im weiteren zeitlichen Verlauf rückten nicht-relationale Ansätze und Technologien mehr und mehr in den Hintergrund.

Der Begriff *NoSQL* wurde erstmals von Carlo Strozzi verwendet, um das Konzept seines Datenbankprojektes hervorzuheben, welches die Daten nicht in Relationen speichert und auch keine SQL-Schnittstelle aufweist. Insofern sollte man hier besser von nicht-relationalen Datenbanken („NoREL“) sprechen.[2] Außerhalb wissenschaftlicher Kreise erfuhren NoSQL-Datenbanken erstmals im Zusammenhang mit massiv skalierbaren Internetanwendungen technologisch an Bedeutung, als riesige Datenmengen (Big Data) hochverfügbar und parallel verarbeitet werden mussten und RDBMS hierfür ungeeignet erschienen. Daher ist die Entstehung und Entwicklung von NoSQL-Datenbanken in den 2000er Jahren eng mit der Entwicklung der Suchmaschine Google und zugehöriger Anwendungen verknüpft.[3]

2.1.1 Definition: NoSQL und NoSQL-Datenbanksystem

Es existiert zwar keine allgemein anerkannte Definition des Begriffs NoSQL, jedoch wird nicht selten hierunter die Abkürzung für „Not only SQL“ verstanden. Im Folgenden sollen unter dem Begriff NoSQL alle nicht-relationalen Ansätze im Datenmanagement verstanden werden, bei denen die Daten nicht in Form von Tabellen gespeichert werden und die Abfragesprache nicht SQL ist.[4]

Eine *NoSQL-Datenbank* lässt sich gut im Kontrast zu einer relationalen Datenbank anhand von fünf Eigenschaften definieren, wie in Tabelle 2.1 dargestellt.[5] Darüber hinaus muss ein NoSQL-Datenbanksystem per Definition den Anforderungen bezüglich umfangreicher Datenbestände (Volume), flexibler Datenstrukturen (Variety) und Echtzeitverarbeitung (Velocity) genügen.[6]

---

[1] Vgl. [Tiw11, S. 4]

[2] Vgl. [SGR15], dort zitiert aus *Carlo Strozzi: NoSQL - A Relational Database Management System;* `http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/NoSQL/Home%20Page` (besucht am 29.11.2016)

[3] Vgl. [Tiw11, S. 4ff]

[4] Vgl. [MK16, S. 18]

[5] Vgl. [SGR15], [MK16, S. 10f]

[6] Vgl. [MK16, S. 20]

	NoSQL-Datenbank	**Relationale Datenbank**
Datenmodell	nicht-relational	relational
Datenschema	ohne fixes Schema	Definition der Tabellen und ihrer Merkmale
Zugriff	Schnittstelle (API) für die Nutzung in Anwendungen	SQL für Definition, Selektion und Manipulation von Daten
Architektur	Schwerpunkt auf horizontale Skalierbarkeit, einfache Datenreplikation über dezentrale, massiv verteilte Architektur	Schwerpunkt auf Datenunabhängigkeit, Entkopplung von Daten und Anwendungsprogrammen
Datenintegrität	zeitlich verzögerte Datenintegrität gewährleistet	Bereitstellung von Hilfsmitteln zur Datenintegrität

Tabelle 2.1: Abgrenzende Eigenschaften von NoSQL- und relationalen Datenbanken

2.1.2 Anforderungen an NoSQL-Datenbanksysteme

Die Entscheidung für ein nicht-relationales Datenbanksystem erwächst häufig daraus, dass die Abbildung von Daten in einem Relationenmodell als Zeilen und Spalten einer Tabelle inadäquat sind. Insbesondere die normalisierte Form im Relationenmodell erzeugt einen Mehraufwand bei der Datenverarbeitung, weil über mehrere Tabellen hinweg Daten verknüpft werden müssen und auch rekursive Abfragen sehr umständlich abzubilden sind. Ferner impliziert ein relationales Datenmodell eine vertikal und horizontal homogene Datenstruktur. Unter einer horizontal homogenen Datenstruktur versteht man, dass die Zeile einer Tabelle ein fixes und jeweils gleiches Format aus einzelnen Spalten haben. Bei einer vertikal homogenen Datenstruktur besitzen die Werte einer Spalte den selben Wertebereich und bestehen aus primitiven Datentypen, die das Abbilden komplexer Strukturen nahezu ausschließen.[7]

Die Anforderungen an nicht-relationale Datenbanksysteme für das Datenmanagement sind durch den typischen Aufbau einer Datenbank und anwendungsspezifische Gesichtspunkte geprägt.[8]

- **Datenstrukturen:** Fachliche Domänen, wie etwa soziale Netzwerke und das Semantic Web, bilden ihre komplexen Datenstrukturen innerhalb eines Graphenmodells ab. Datenmodelle für Geoinformationssysteme (GIS) und Anwendungen im Bereich des Computer Aided Design (CAD) bestehen aus unzähligen komplexen Substrukturen.

- **Unabhängigkeit vom Datenschema:** Hierunter versteht man flexible Datenstrukturen, denen keine Definition des Datenschemas zugrunde liegt. Dieser Umstand ermöglicht auch Daten den selben Wert in verschiedenen Arten zu repräsentieren (beispielsweise als Zeichenkette oder numerischer Wert) und zu verarbeiten.

- **Selbstbeschreibungsfähigkeit:** Als Konsequenz aus den Anforderungen komplexer Datenstrukturen und der Unabhängigkeit vom Datenschema ergibt sich Variabilität, die konstante Veränderung von Daten sowie die Selbstbeschreibungsfähigkeit ihrer Eigenschaften und Werte. Dies kann auch in Form von Metadaten geschehen.

[7] Vgl. [Wie15, S. 34ff]
[8] Vgl. [Wie15, S. 37f]

Datenmenge als 10er-Potenz	Datenmenge als 2er-Potenz
Kilobyte (kB) – 10^3	Kibibyte (KiB) – 2^{10}
Megabyte (MB) – 10^6	Mebibyte (MiB) – 2^{20}
Gigabyte (GB) – 10^9	Gibibyte (GiB) – 2^{30}
Terabyte (TB) – 10^{12}	Tebibyte (TiB) – 2^{40}
Petabyte (PB) – 10^{15}	Pebibyte (PiB) – 2^{50}
Exabyte (EB) – 10^{18}	Exbibyte (EiB) – 2^{60}
Zettabyte (ZB) – 10^{21}	Zebibyte (ZiB) – 2^{70}
Yottabyte (YB) – 10^{24}	Yobibyte (YiB) – 2^{80}

Tabelle 2.2: Größenordnungen Big Data

- **Big Data:** Mit dem häufig im Kontext von NoSQL verwendeten Begriff Big Data werden riesige Datenmengen (im größeren Terabytebereich, siehe Tabelle 2.2[9]) verstanden, welche mit dem Konzept relationaler Datenbanken nur eine unbefriedigende Handhabung ermöglichen. Laut Meier und Kaufmann [MK16] sind trotz fehlender Definition folgende drei „V“ kennzeichnend für Big Data: *„Volume (umfangreicher Datenbestand), Variety (Vielfalt von Datenformaten; strukturierte, semi-strukturierte und unstrukturierte Daten) und Velocity (hohe Verarbeitungsgeschwindigkeit und Echtzeitverarbeitung).“*[10]

- **Skalierbarkeit:** Ein Datenbanksystem soll mit massiv verteilten Datenbeständen, die über Knoten (Servern) verbunden sind, umgehen können. Darüber hinaus ist die Fähigkeit zur dynamischen horizontalen Skalierung erforderlich, so dass Knoten unterbrechungsfrei zu einem Cluster hinzugefügt und daraus entfernt werden können. Dies umfasst ebenfalls den transparenten Zugriff auf das Datenbanksystem ohne Kenntnis der Knoten.

- **Verarbeitungsgeschwindigkeit:** Umfangreiche Datenbestände sollen mit einem hohen Datendurchsatz gelesen und geschrieben werden, um eine Echtzeitverarbeitung bei hoher Verfügbarkeit und akzeptabler Antwortzeit zu ermöglichen.

2.1.3 Konsistenzmodell

Der Umgang mit der Datenintegrität, sprich der Konsistenz der Daten, unterscheidet sich bei NoSQL-Datenbanken vom bekannten *ACID-Prinzip* relationaler Datenbanken. Dieses ACID-Prinzip stellt sicher, dass eine Transaktion folgende Bedingungen erfüllt:[11]

- Atomarität (Atomicity) heißt, dass eine Transaktion, also eine Abfolge von Datenoperationen in verschiedenen Tabellen und Zeilen, entweder ganz oder gar nicht ausgeführt wird.

- Konsistenz (Consistency) bezeichnet die Widerspruchsfreiheit der Daten innerhalb einer Transaktion beim Übergang zwischen zwei konsistenten Zuständen.

- Isolation bedeutet, dass gleichzeitig ablaufende Transaktionen dasselbe Ergebnis liefern wie Transaktionen in einer Einzelnutzerumgebung liefern.

[9] Tabelle angelehnt an [Tiw11, S. 7]
[10] Zitiert aus [MK16, S. 11]
[11] Vgl. [Tiw11, S. 169ff]

- Dauerhaftigkeit (Durability) meint, dass der Zustand nach einer erfolgreich abgeschlossenen Transaktion so lange erhalten bleibt, bis er von der nächsten Transaktion geändert wird.

Aufgrund der Anforderungen bezüglich Skalierbarkeit und Verarbeitungsgeschwindigkeit erfolgt bei NoSQL-Datenbanken die Gewährleistung der Datenintegrität nach dem *BASE-Prinzip*, einer weichen Konsistenzforderung (Weak Consistency). Das Akronym BASE steht für Basically Available, Soft State, Eventually Consistent. Es besagt, dass eine Änderung von Daten durch eine Operation in einem massiv verteilten Datenbanksystem den anderen Knoten im System mitgeteilt wird. Hierbei kommt es zu einer zeitlichen Latenz, so dass andere Knoten nicht den aktuellen Zustand anzeigen und erst zeitlich verzögert die Änderung vollziehen. Einzelne Knoten im System sind meistens verfügbar (Basically Available), haben jedoch nicht immer einen konsistenten Datenstand (Eventually Consistent) und können sich daher in einem weichen Zustand befinden (Soft State). Die Tabelle 2.3 zeigt weitere Unterschiede zwischen dem BASE- und ACID-Prinzip auf.[12]

BASE-Prinzip	**ACID-Prinzip**
Konsistenz wird verzögert weitergegeben (Weak Consistency)	Konsistenz hat oberste Priorität (Strong Consistency)
Verwendung optimistischer Synchronisationsverfahren mit Differenzierungsoptionen	Verwendung pessimistischer Synchronisationsverfahren mit Sperrprotokollen
Hohe Verfügbarkeit bzw. Ausfalltoleranz bei massiv verteilter Datenmenge	Verfügbarkeit bei üblichen Datenmengen gegeben

Tabelle 2.3: BASE- und ACID-Prinzip im Vergleich

2.1.4 Brewer's CAP-Theorem

Das *CAP-Theorem* wurde erstmals im Jahr 2000 durch Eric Brewer als Vermutung artikuliert und später von Gilbert und Lynch bewiesen: In einem massiv verteilten Datenbanksystem können die drei Eigenschaften Konsistenz (Consistency), Verfügbarkeit (Availability) und Ausfalltoleranz (Partition Tolerance) nicht gleichzeitig gelten.[13] Es können nur jeweils zwei der drei Eigenschaften gelten (dargestellt in Abbildung 2.1[14]).

Der Begriff der Konsistenz ist hierbei nicht zu verwechseln mit demjenigen des ACID-Prinzips. Konsistenz meint hierbei die Tatsache, dass in einem massiv verteilten System alle Knoten und ihre Daten zu jedem Zeitpunkt innerhalb konkurrierender Transaktionen denselben Zustand haben sollen.[15] Verfügbarkeit bezeichnet den ununterbrochenen Betrieb des Datenbanksystems und umfasst wohldefinierte Antwortzeiten.[16] Mit Ausfalltoleranz wird die Fähigkeit eines Systems bemessen, jederzeit unterbrechungsfrei betrieben werden zu können, selbst wenn Knoten ausfallen, in das System hinzugefügt oder entfernt werden.[17]

[12] Vgl. [MK16, S. 154f]
[13] Vgl. [GL02]
[14] Linke Abbildung entnommen aus [MK16, S. 149]
[15] Vgl. [Tiw11, S. 174]
[16] Vgl. [MK16, S. 148ff]
[17] Vgl. [Tiw11, S. 175]

Abbildung 2.1: Kombinationen im CAP-Theorem

2.1.5 Typen von NoSQL-Datenbanken

Es werden in der allgemeinen Literatur vier Typen von NoSQL-Datenbanken unterschieden[18] (siehe hierzu auch Abbildung 2.2[19]):

- **Schlüssel-Wert-Datenbank (Key-Value Database):** Hierbei handelt es sich um das einfachste mögliche Modell einer Datenbank, wobei die Daten in Form eines Tupels angelegt sind und aus einem eindeutigen Schlüssel (Key) und einem Wert (Value) bestehen. Dieses Konzept lässt sich um Namensräume für die Schlüssel erweitern, findet jedoch seine Beschränkung in der Datenstruktur, welche keinerlei Verschachtelungen oder Referenzen zulässt.[20] Dieses schemafreie Konzept ist als Hash Table bekannt und eignet sich besonders für große Datenmengen, da es beliebig skalierbar ist mittels Sharding, sprich der Datenfragmentierung auf einzelnen Knoten respektive Shrads,.[21]

- **Spaltenfamilien-Datenbank (Column-Oriented Database):** Bei diesem Datenbankmodell werden Daten nicht zeilenweise (wie in einem RDBMS), sondern spaltenweise geschrieben. Diesem Vorgehen liegt die Idee eines optimierten Lesezugriffs zugrunde, weil beim Lesen einer Datenzeile oft nicht alle Spalten benötigten werden. Gleichzeitig existieren bestimmte Gruppen von Spalten, die oftmals gemeinsam gelesen werden. Für einen effizienten Lesezugriff werden nun solche Gruppen von Spalten, sogenannte Spaltenfamilien, als Speichereinheit aufgefasst.[22] Spaltenfamilien-Datenbanken können als eine Erweiterung von Schlüssel-Wert-Datenbanken angesehen werden, sie sind jedoch nicht schemafrei. Außerdem bieten sie aufgrund der Datenhomogenität innerhalb der Spalten (gleicher Datentyp, Wertebereich) die Vorteile einer Lauflängenkompression und weitere Encodierungsmöglichkeiten der Spaltenfamilien an. Als ineffizient und nachteilig zu bezeichnen sind bei diesem Datenbankkonzept das Schreiben der Datenzeilen respektive Spalten sowie das Lesen (Rekonstruktion) einer Zeile.[23]

- **Dokumentdatenbank (Document Database):** Als Dokument soll bei diesem Datenbankmodell ein strukturierter Datensatz (beispielsweise im XML- oder JSON-Format) verstanden werden. Eine Dokumentdatenbank verbindet das Prinzip und die Vorteile einer Schlüssel-Wert-Datenbank mit den Möglichkeiten einer komplexen Datenstruktur (in Form von Arrays, Datentypen, Restriktionen und Schemas) für

[18] Vgl. [ABF14b], [SGR15] u.a.
[19] Abbildung entnommen aus [MK16, S. 19, 228]
[20] Vgl. [Wie15, S. 105f]
[21] Vgl. [MK16, S. 224f]
[22] Vgl. [MK16, S. 226ff]
[23] Vgl. [Wie15, S. 143ff]

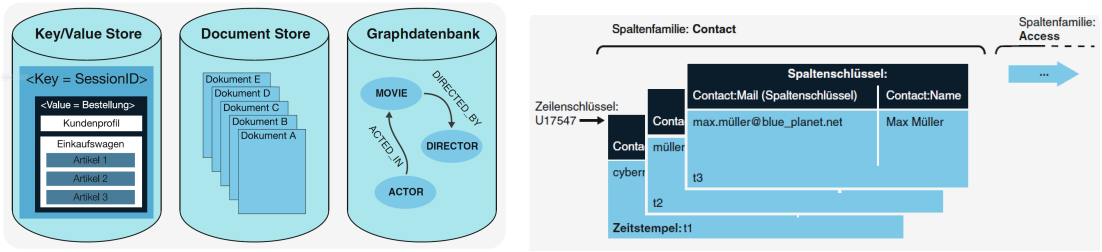


Abbildung 2.2: Unterschiedliche Typen von NoSQL-Datenbanken

den Wert eines Schlüssels.[24] Trotz der Datenstrukturen sind Dokument-Datenbanken als völlig schemafrei anzusehen sind, weil es vor dem Einfügen von Daten nicht notwendigerweise einer Datenstruktur bedarf. Der Umgang mit der Datenstruktur liegt somit beim Benutzer.[25]

- **Graphdatenbank (Graph Database):** Im Graphenmodell, auf welchem die Graphdatenbanken basieren, repräsentieren die Knoten im Graphen die Daten oder die Struktur und die Kanten stellen die Beziehung der Knoten untereinander dar. Dieses Datenmodell ist besonders für den Fall geeignet, wenn Daten in Netzwerken organisiert sind, die Verbindung zwischen den Datenobjekten eine herausragende Rolle einnimmt und es dafür einer eigenen Semantik bedarf.[26] Eine Graphdatenbank unterstützt – ähnlich wie relationale Datenbanken – ein Konzept zur Datenintegrität bezogen auf die Strukturen des Graphen, wie zum Beispiel Knoten und Kanten, Attribute und Werte sowie referenzielle Integrität der Kanten. Einige Implementierungen von Graphdatenbanken unterstützen das ACID-Prinzip. Der Nachteil einer Graphdatenbank besteht aufgrund der immanenten Eigenschaft der Abhängigkeiten zwischen den Knoten und in der komplexen Datenfragmentierung für eine mögliche Skalierung. Sharding wird daher nicht unterstützt.[27]

2.2 Systemqualitäten

NoSQL-Datenbanken werden üblicherweise für datenintensive Anwendungen genutzt, die sich unterscheiden lassen in OLTP- und OLAP-Anwendungen. OLTP-Systeme (Online Transaction Processing) bewerkstelligen Geschäftsprozesse in Echtzeit, was typischerweise einhergeht mit weiteren Anforderungen, wie zum Beispiel die schnelle Verarbeitung vieler Transaktionen unter Beachtung einer strengen Datenintegrität. Im Gegensatz hierzu werden OLAP-Systeme (Online Analytical Processing) für langlaufende Auswertungen größerer Datenmengen genutzt. Da es sich um aggregierte und konsolidierte Daten handelt, gelten hierbei weichere Anforderungen an die Datenintegrität.[28] Ein weiteres Anwendungsfeld ist der technische Betrieb von sozialen Medien und die daraus resultierenden Anforderungen hinsichtlich der zu verarbeitenden Datenmengen und der Ausführungsgeschwindigkeit.[29]

[24] Vgl. [Wie15, S. 109ff]
[25] Vgl. [MK16, S. 228ff]
[26] Vgl. [Wie15, S. 41ff]
[27] Vgl. [MK16, S. 238f]
[28] Vgl. [Sak16]
[29] Vgl. [SE16]

2.2.1 Cloud Computing

Die Anforderung der Skalierbarkeit (siehe Abschnitt 2.1.2) und die technologische Entwicklung der vergangenen Jahre haben dazu geführt, dass NoSQL-Datenbanken zu den Cloud-Systemen gezählt werden.[30] Das amerikanische National Institute of Standards and Technology definiert den Begriff *Cloud Computing* wie folgt:

> „Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.“[31]

NoSQL-Datenbanken als spezielle Form eines Cloud Service verfügen über Mandantenfähigkeit. Es wird hierbei zwischen drei Ansätzen zur Implementierung der Mandantenfähigkeit bei Datenbanksystemen unterschieden: Bei Shared Server teilen sich die Mandanten einen Datenbankserver und besitzen jeweils eigene Datenbanken. Bei Shared Process verfügen die Mandanten über eigene Tabellen innerhalb einer geteilten Datenbank. Bei Shared Table teilen sich die Mandanten eine Tabelle, nur die jeweiligen Zeilen beziehungsweise Spalten sind mandantenspezifisch.[32]

2.2.2 Datenmodelle

Die Entscheidung für eine NoSQL-Datenbank aus der großen Auswahl nicht-relationaler Implementierungen[33] erfolgt zuvorderst unter dem Aspekt des Datenmodells, welches von der einzusetzenden Datenbank unterstützt werden soll. Eine qualitative Analyse des intendierten Datenmodells bezüglich der Datenbanktypen ermöglicht es, Kandidaten auszuwählen.[34] Weitere Aspekte, die bei einer qualitativen Analyse berücksichtigt werden, sind die Anforderungen für die Auswahl einer NoSQL-Datenbank sind die Möglichkeiten eine Abfragesprache zur Datenanalyse (Ad-hoc Query) einzusetzen, weitere Werkzeuge zur Verwaltung der Datenbank (Management Tools) und die Fähigkeiten des Systems zur Indizierung von Daten.[35] Letztlich ausschlaggebend für die Wahl eines NoSQL-Datenbanksystems ist die Komplexität des zugrundeliegenden Datenmodells, die Implementierung dessen in der Anwendung sowie die zu erwartende Leistung in Form von Verarbeitungsgeschwindigkeit beim Lesen und Schreiben der Daten.[36]

Mit dem Begriff *polyglotte Persistenz (Polyglot Persistence)* werden Anwendungen beschrieben, die zwei oder mehrere unterschiedliche Datenmodelle und Typen von Datenbanken nutzen, wenn ein Datenmodell allein nicht allen Anforderungen gerecht werden kann. Die Vorteile einer solchen Lösung liegen in der besseren Verarbeitungsgeschwindigkeit. Allerdings ist auch mit einer Reihe von Nachteilen zu rechnen: Häufig müssen Daten zwischen den einzelnen Datenmodellen und Datenbanktypen abgebildet und synchronisiert werden, was einen Mehraufwand erzeugt. Ebenso kann die Handhabung der unterschiedlichen Schnittstellen der einzelnen Datenbanksysteme und die Umsetzung von Transaktionen mühsam sein. Eine Lösung hierfür können NoSQL-Datenbanksysteme darstellen, die mehrere Datenmodelle unterstützen, sogenannte *Multi-Modell-Systeme (NoSQL*

[30] Vgl. [Coo+10]
[31] Zitiert aus [MG11]
[32] Vgl. [Sak16]
[33] Die Plattform `http://nosql-database.org/` listet aktuell 225 unterschiedliche NoSQL-Datenbanksysteme auf (besucht am 5. Dezember 2016).
[34] Vgl. [Coo+10]
[35] Vgl. [LM13]
[36] Vgl. [Fio+16]

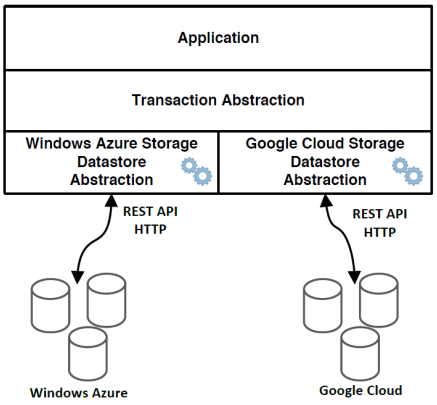


Abbildung 2.3: Implementierung eines transaktionalen Zugriffsprotokolls

Multi-Model Systems). Der Nutzen einer einfacheren Softwareentwicklung geht dabei einher mit dem Verlust von Leistung bei der Datenverarbeitung.[37]

2.2.3 Transaktionen

Eine *Transaktion* wird definiert als Abfolge von Operationen, die atomar, konsistent, isoliert und dauerhaft ist. Die Verwaltung von Transktionen ermöglicht ein konfliktfreies Arbeiten bei konkurrierenden Datenzugriffen.[38] Üblicherweise besitzen NoSQL-Datenbanken keinen Mechanismus zur Ausführung transaktionaler Operationen. Wenn jedoch Transaktionen unterstützt werden, dann in der Form, dass ein Datensatz gesperrt wird. Es ist dann keine weitere Bearbeitung möglich, wobei dieser Datensatz weiterhin gelesen werden kann.[39]

Es werden drei Möglichkeiten zur Implementierung der Transaktionalität unterschieden: Zum einen lässt sich ein Mechanismus auf Datenbankebene realisieren, was aber äußerst komplex ist und die Skalierbarkeit und Verfügbarkeit des Systems mindert. Eine andere Möglichkeit ist die Nutzung von Middleware, die geeignet sind für Anwendungen, die in einer kontrollierten Umgebung ausgerollt sind. Diese Art der Implementierung ist ebenfalls als komplex zu klassifizieren. Eine dritte Möglichkeit besteht in der Definition eines transaktionalen Zugriffsprotokolls, durch welches jene Anwendungen, die auf die Datenbanken zugreifen um Datenoperationen auszuführen. Der Vorteil dieses Ansatzes besteht darin, dass dieser frei von Einflüssen auf die Skalierbarkeit und Verfügbarkeit des Datenbanksystems ist.[40]

Es existieren bereits einige experimentelle Systeme, die eine Transaktionsunterstützung über mehrere Datensätze hinweg in verteilten, heterogenen Schlüssel-Wert-Datenbanken ermöglichen. Dabei wird ein transaktionales Zugriffsprotokoll in Gestalt einer Schnittstelle verwendet. Diese operiert zwischen der Anwendung, welche auf die Daten zugreift, und der Datenbankimplementierung, welche diese Daten bereitstellt. Dies ist in Abbildung 2.3[41] dargestellt. Das Verfahren zur Umsetzung einer Transaktion beruht darauf, dass ein Datensatz um Kopfdaten, wie zum Beispiel Zeitstempel, erweitert wird, sobald er gelesen oder bearbeitet wird.[42]

[37] Vgl. [OVC16]
[38] Vgl. [MK16, S. 244]
[39] Vgl. [Sak16]
[40] Vgl. [DFR15]
[41] Abbildung entnommen aus [DFR15]
[42] Vgl. [DFR15]

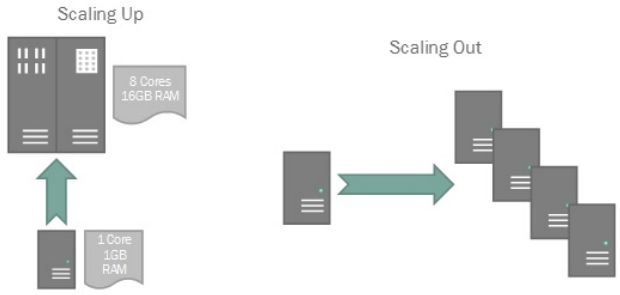


Abbildung 2.4: Vertikale (Scale Up) und horizontale (Scale Out) Skalierung

2.2.4 Skalierbarkeit und Elastizität

Im Allgemeinen versteht man unter *Skalierbarkeit* ein Konzept, bei dem ein System, Netzwerk oder Prozess fähig ist, unter Hinzufügen weiterer Ressourcen die Leistung zu erhöhen. Dies kann in Form von vertikaler oder horizontaler Skalierung geschehen. Als *vertikale Skalierung (Scale Up)* bezeichnet man das Hinzufügen von Ressourcen zum Knoten eines Systems. *Horizontale Skalierung (Scale Out)* bedeutet das Hinzufügen weiterer Knoten zum System (siehe Abbildung 2.4).

Nach Bondi lassen sich vier Arten von Skalierbarkeit unterscheiden:[43]

- **Lastskalierbarkeit** bezeichnet die Eigenschaft bei der ein System unter leichter, moderater oder starker Last ohne spürbare Verzögerungen bei der Verarbeitung und weiterhin effizienter Nutzung der Systemressourcen weiter arbeitet.
- **Räumliche Skalierbarkeit** liegt dann vor, wenn der Speicherverbrauch eines Systems bei steigender Anzahl zu verwaltender Elemente höchstens sublinear ansteigt (siehe Abbildung 2.5(b)).[44]
- **Räumlich-zeitliche Skalierbarkeit** liegt dann vor, wenn die Menge der Objekte in einem System sich nicht unmittelbar auf die Leistungsfähigkeit auswirkt.
- **Strukturelle Skalierbarkeit** meint die Eigenschaft eines Systems, dass es sich nicht aufgrund der eigenen Struktur bei der Skalierung behindert.[45]

Massiv verteilte NoSQL-Datenbanken sind in der Lage, mit großen Datenmengen umzugehen und viele Anfragen zu bearbeiten. Dies geschieht, indem die verschiedenen Anfragen über eine Lastverteilung auf die zur Verfügung stehenden Instanzen des Systems verteilt werden. Eine Instanz hält immer nur einen Teil aller Daten. Ein horizontal skalierbares (Scale-out) NoSQL-Datenbanksystem soll daher alle Daten und Anfragen so über die Instanzen verteilen, dass keine Engpässe bei der Verarbeitung entstehen.[46]

Unter dem Begriff *Elastizität (Elasticity)* versteht man das Hinzufügen weiterer Ressourcen in Form von Leistung zu den einzelnen Knoten.[47] Damit wird das Konzept der vertikalen Skalierung aufgegriffen und um den Aspekt des unterbrechungsfreien Betriebs beim Hinzufügen oder Entfernen der Ressourcen erweitert. Darüber hinaus wird von einem im Cloud Computing betriebenen NoSQL-Datenbanksystem eine hohe Verfügbarkeit

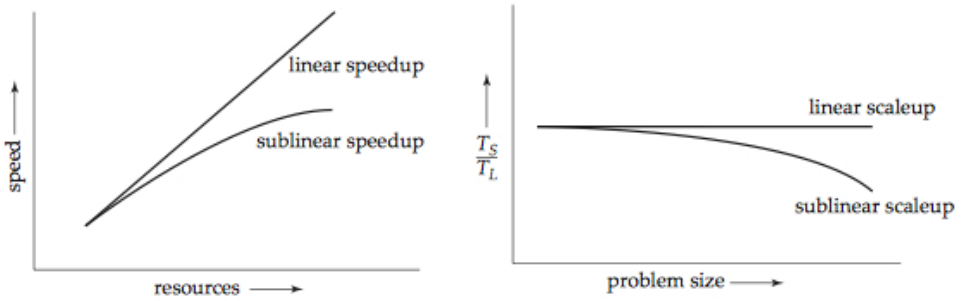
[43] Vgl. [Bon00]

[44] Dies bedeutet, dass mit zunehmender Anzahl Objekte der Speicherverbrauch nur durch die Objekte selbst und nicht durch die Verwaltung dieser Objekte ansteigt.

[45] Beispiel: Aufgrund der Größe eines Adressbereiches (angenommen: 0–255) kann nur eine bestimmte Anzahl Objekte verwaltet werden.

[46] Vgl. [Dey+14]

[47] Vgl. [Coo+10]



(a) Speed Up (b) Scale Up

Abbildung 2.5: Speed Up und Scale Up mit steigender Problemgröße

erwartet. Dies wird erreicht durch den Betrieb auf relativ günstiger und austauschbarer Standard-Hardware (Commodity Hardware).[48]

Der *Beschleunigungsgrad (Speed Up)* beschreibt den Zusammenhang bezüglich der Verarbeitungsgeschwindigkeit für einen vorgegebenen Task unter Zuhilfenahme weiterer Ressourcen, wie Prozessoren und Speicher, wie Abbildung 2.5(a)[49] zeigt. Der Beschleunigungsgrad verläuft linear, wenn die Verarbeitungsgeschwindigkeit sich antiproportional zu den allozierten Ressourcen verhält. Der Grad der *vertikalen Skalierung (Scale Up)* verläuft linear, wenn das Verhältnis der Verarbeitungsgeschwindigkeit mit ansteigender Problemgröße unter Hinzunahme weiterer Ressourcen gleich bleibt (siehe Abbildung 2.5(b)[50]).

2.3 Systemklassifikationen

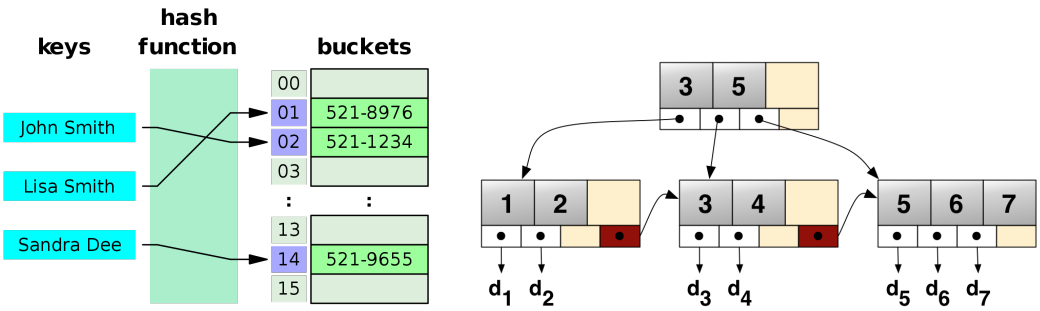
Eine NoSQL-Datenbank wird typischerweise als Cloud Service in einem Cluster betrieben, welches ein Verbund mehrerer Knoten, sprich einzelner Betriebssysteminstanzen, darstellt. Diese Systemarchitektur ermöglicht die Umsetzung der in Abschnitt 2.2.4 vorgestellten Konzepte zur Skalierbarkeit, Ausfalltoleranz und auch Lastverteilung der Datenbasis.

Die Entscheidung für ein bestimmtes NoSQL-Datenbanksystem ist – wie bei allen Computersystemen – abhängig von den Anforderungen, mit denen die intendierten Systemeigenschaften implementiert werden sollen. Weil es kein ein NoSQL-Datenbanksystem gibt, welches allen Anforderungen gerecht wird, muss zumeist ein Kompromiss (Tradeoff) innerhalb dieser Faktoren gesucht werden.[51] In den folgenden Abschnitten werden diese Faktoren in Form von Systemklassifikationen beschrieben.

2.3.1 Schreib- und Lesegeschwindigkeit

Die Leistungsfähigkeit bezüglich der Schreib- und Lesegeschwindigkeit eines Datenbanksystems hängt hauptsächlich von der Architektur des zugrundeliegenden Speichersystems ab (siehe Speicherpyramide in Abbildung 2.7). Im Unterschied zu herkömmlichen RDBMS, welche auf zentral verwalteten Speicher-Arrays basieren und für diese Art der Benutzung optimiert sind, verwenden NoSQL-Datenbanksysteme individuelle Speichermedien, wie beispielsweise Flash-Speicher für latenzarme Schreib- und Leseoperationen

[48] Vgl. [Coo+10], [Dey+14]
[49] Abbildung entnommen aus [SKS11, S. 778]
[50] Abbildung entnommen aus [SKS11, S. 779]
[51] Vgl. [Coo+10], [Dey+14]



(a) Hashtabelle (b) B⁺-Baum

Abbildung 2.6: Funltionsweise einer Hashtabelle und eines B⁺-Baums

im Bereich von Nano- und Mikrosekunden. Diese Speichermedien werden – anders als bei einem relationalen System – nicht durch das Datenbanksystem selbst verwaltet, sondern vom Betriebssystem, mit dem die Datenbank betrieben wird. Die Verfügbarkeit respektive Ausfallsicherheit wird durch inhärente Eigenschaften der Datenbank und der von ihr genutzten Protokolle gewährleistet.[52]

Beim Datenzugriff wird unabhängig vom Typ des Datenbanksystems unterschieden zwischen *Point Queries*, der assoziativen Suche nach einem Wert für ein bestimmtes Attribut (Beispiel: `name="Bob"`), und *Range Queries*, der Suche nach einem Attribut innerhalb eines bestimmten Wertebereiches (Beispiel: `20 < Temperatur < 30`). Zusätzlich hierzu können RDBMS eine gesamte Tabelle für den Datenzugriff einlesen (Entire Relation Scan).[53] Im Gegensatz zu zeilen- oder spaltenbasierten Datenbanksystemen sind nichtrelationale Datenbanksysteme darauf ausgelegt, große Datenmengen zu schreiben und mittels Point oder Range Queries zu durchsuchen und zu lesen.[54]

Mit Ausnahme der Spaltenfamilien-Datenbanken ist die logische Organisation der Daten bei allen NoSQL-Datenbanken ähnlich: Einem vom System generierten Schlüssel ist ein Wert – je nach Datenmodell auch Objekt, Dokument oder eine Struktur – zugewiesen. Die Datenbank strukturiert und verteilt diese Datenmengen gleichmäßig auf die zur Verfügung stehenden Knoten. Hierfür werden Hashtabelle (Funktionsweise dargestellt in Abbildung 2.6(a)[55]) oder Varianten von B-Bäumen (B⁺-Baum schematisch dargestellt in Abbildung 2.6(b)[56]) genutzt. Eine Hashtabelle eignet sich besonders für Random I/O. Unter *Random I/O* versteht man den Umstand, dass nicht bekannt ist, welcher Datensatz als nächstes gelesen oder geschrieben wird. Dies ist der Fall bei Point Queries sowie Insert- und Update-Operationen. Bei der Verwendung von B-Bäumen werden hingegen die Daten sequenziell geschrieben und gelesen, so dass zum Auffinden eines gesuchten Schlüssel-Wert-Paars die Datenstruktur traversiert werden muss.[57]

Der physikalische Schreib- und Lesezugriff bei NoSQL-Datenbanken ähnelt denen relationaler Datenbanken. Es wird ein Log-basiertes Verfahren verwendet, bei denen die Daten in einem Log sequenziell angehängt werden (Sequential/Write-Ahead Logging). Das bedeutet bei einem Insert oder Update, dass die Änderung des Datensatzes am Ende des

[52] Vgl. [Sch12]
[53] Vgl. [SKS11, S. 799]
[54] Vgl. [Sch13]
[55] Abbildung entnommen von `https://commons.wikimedia.org/wiki/File:Hash_table_3_1_1_0_1_0_0_SP.svg`
[56] Abbildung entnommen von `https://de.wikipedia.org/wiki/Datei:Bplustree.png`
[57] Vgl. [Coo+10], [Sch12], [Sch13]

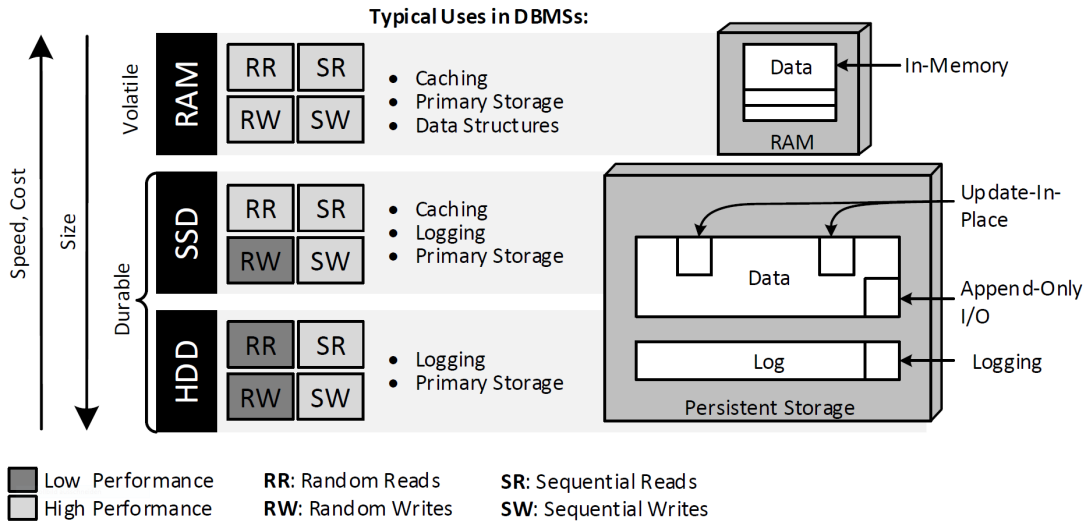


Abbildung 2.7: Speicherpyramide

Logs geschrieben wird. Der Vorteil dieses Ansatzes (Append-Only I/O) liegt in einem effizienten Schreiben eines Datensatzes, weil die Daten einfach nur angehängt werden (Optimized for Writes). Dieses Prinzip bedingt allerdings auch, dass Datensätze, die sich über die Zeit geändert haben, besonders zeitintensiv wiederherzustellen (lesen) sind. Der zeitliche Nachteil lässt sich dadurch abmildern, dass beim Update nicht einfach der geänderte Wert an das Log angehängt wird, sondern der ganze Datensatz neu geschrieben und der vorherige invalidiert wird. Dies stellt ein Kompromiss zur Variante *Update-In-Place* dar, bei dem der geänderte Wert an der bereits existierenden Stelle im Log überschrieben wird (Optimized for Reads). Der zeitliche Nachteil liegt hier beim Schreiben des Datensatzes.[58]

Die Abbildung 2.7[59] zeigt die Ergebnisse der Studie von Gessert et al. [Ges+16] zu Umsetzungskosten (Costs) und zur Ausführungsgeschwindigkeit (Speed) in NoSQL-Datenbanken bezüglich der Nutzung verschiedenener Speichermedien: RAM (Random-Access Memory; Arbeitsspeicher), SSD (Solid-State Drive; Flash-Speicher) und HDD (Hard Disk Drive; magnetischer Speicher). Die Autoren untersuchten die unterschiedlichen Leistungseigenschaften in Bezug zum Speichermedium für die im vorherigen Abschnitt vorgestellten Verfahren zur Datenorganisation sowie zum Schreib- und Lesezugriff. Ferner verdeutlichen die Autoren, dass die Speicherarchitektur einer Datenbank eine räumliche Dimension (Speicherort: Update-In-Place versus Append-Only I/O) und eine temporale Dimension (Speicherzeitpunkt: volatil versus dauerhaft; RAM versus SSD/HDD) hat. In der Kombination dieser verschiedenen Speichermanagement-Strategien wird die Stärke und der Grund für die Diversität von NoSQL-Datenbanksystemen gesehen.[60] Die Ergebnisse der Studie von Gessert et al. [Ges+16] zum Leistungsverhalten der unterschiedlichen Speichermedien sind deckungsgleich zu den Aussagen von Schindler [Sch13] hierzu.

2.3.2 Latenz und Dauerhaftigkeit

Eine weitere Kategorie der Systemklassifikation, die im Zusammenhang mit der Schreibgeschwindigkeit eines NoSQL-Datenbanksystems steht, ist die Latenz respektive Dauerhaftigkeit eines geschriebenen Datensatzes: Datensätze können zuerst in den RAM geschrieben und erst später auf SSD/HDD synchronisiert und dauerhaft persistiert werden.

[58] Vgl. [Coo+10], [Sch12], [Sch13]
[59] Abbildung entnommen aus [Ges+16]
[60] Vgl. [Ges+16]

Die Vorteile von diesem Verfahren liegen in einer niedrigen Latenz beim Schreiben der Daten mit einer höheren Verarbeitungsgeschwindigkeit durch den schnelleren, jedoch volatilen Arbeitsspeicher sowie der späteren Synchronisierung der Daten (siehe Abschnitt 2.3.3) in Intervallen auf der verwendeten Festplatte. Nachteilig ist dieses Prinzip, wenn es zu einem Systemausfall kommt und die Daten im Arbeitsspeicher verloren gehen, weil sie nicht auf Festplatte persistiert werden konnten. Es gilt hierbei einen Kompromiss zu finden zwischen einer niedrigen Latenz und der Dauerhaftigkeit der persistierten Daten.[61]

2.3.3 Synchrone und asynchrone Replikation

Die Replikation der von der Datenbank gehaltenen Daten dient der Umsetzung der Skalierbarkeitsanforderungen hinsichtlich der Antwortzeiten, der zu verarbeitenden Datenmengen, der Verfügbarkeit der Knoten und des Datenbanksystems sowie der Wiederherstellung der Daten durch Replikation bei Ausfall eines Knotens.[62] Man unterscheidet hierbei zwischen zwei Ansätzen:

- **Synchrone Replikation** verfolgt das Ziel, alle Kopien eines Knoten auf aktuellem Stand zu halten. Dies führt potenziell zu einer hohen Latenz bei Update-Operationen und zu einer schlechten Verfügbarkeit bei Ausfall eines Knotens.
- **Asynchrone Replikation** ermöglicht hingegen das latenzarme Schreiben der Daten und verhält sich bei Ausfall eines Knotens stabil – zum Nachteil der Dauerhaftigkeit der zu speichernden Daten.

Beide Ansätze zeigen den unmittelbaren Einfluss der Replikationsstrategie auf die Latenz und die Datenintegrität (Konsistenz) in einem massiv verteilten Datenbanksystem.[63] Aus dem CAP-Theorem ist bekannt, dass immer nur zwei der drei Anforderungen bestehend aus Konsistenz (C), Verfügbarkeit (A) und Ausfalltoleranz (P) gelten können. Daher ist ein Kompromiss in den folgenden Varianten (dargestellt in Abbildung 2.1) zu suchen:

- **AP-Systeme** zeigen ein Verhalten, das als verfügbar und ausfalltolerant bezeichnet werden kann. Jedoch ist die Konsistenz der Daten nicht gewährt.
- **CP-Systeme** zeichnen sich durch Datenintegrität und Ausfalltoleranz aus. Dies geht zu Lasten der Verfügbarkeit.
- **CA-Systeme** besitzen ebenfalls eine konsistente Datenbasis und sind verfügbar zu Lasten der Ausfalltoleranz.

In alle diese drei Kategorie fallen relationale Datenbanksysteme.[64]

2.3.4 Datenfragmentierung

Mit dem Begriff *Datenfragmentierung (auch Shrading oder Partitioning)* bezeichnet man in massiv verteilten Datenbanksystemen das Zerlegen und Verteilen der Datenbasis in Teilmengen auf die einzelnen Knoten. Unabhängig vom Typ des Datenbanksystems unterscheidet man zwischen *horizontaler* Datenfragmentierung, bei der unterschiedliche Datensätze auf die verschiedenen Knoten verteilt werden, und *vertikaler* Datenfragmentierung, bei der ein Datensatz anhand seiner Attribute zerlegt und auf die Knoten verteilt wird. Auf dem Prinzip der vertikalen Datenfragmentierung basieren Spaltenfamilien-Datenbanken

[61] Vgl. [Coo+10], [Ges+16], [Sch13]
[62] Vgl. [Coo+10]
[63] Vgl. [Ges+16]
[64] Vgl. [Wie15, S. 306f]

(siehe Abschnitt 2.1.5). Eine Kombination beider Ansätze wird als *hybride* Datenfragmentierung bezeichnet.[65] Üblicherweise werden NoSQL-Datenbanksysteme in einer Shared-Nothing-Architektur betrieben, in der die einzelnen Knoten eigene Prozessoren und Speichermedien besitzen und über ein Netzwerk verbunden sind. Dies steht im Gegensatz zu RDBMS, welche häufig auf einer Shared-Disk-Architektur basieren, bei der die einzelnen Knoten zwar eigene Prozessoren besitzen, aber auf gemeinsame Speichermedien zugreifen.[66]

In Abhängigkeit zum gewählten Datenmodell sind verschiedene Strategien möglich, um Anhaltspunkte zu ermitteln, anhand derer man die Daten horizontal fragmentieren kann. Diese Strategien können auch kombiniert angewendet werden. Hierzu gehören der Typ des Datenzugriffs (hauptsächlich lesend oder schreibend), die Ähnlichkeit der Datenwerte, Zugriffshäufigkeit sowie die Frequenz und die Dauer des Zugriffs. Die Vorteile einer Datenfragmentierung liegen hauptsächlich in der Lastverteilung auf die zur Verfügung stehenden Knoten, in der niedrigeren Netzwerklatenz und in der Verarbeitungsgeschwindigkeit beim Lesen, Schreiben und Ändern von Datensätzen, welche häufig gemeinsam abgefragt werden.[67] Darüber hinaus bietet die vertikale Datenfragmentierung bei spaltenorientierten Datenbanken einen effizienten Zugriff auf Teilmengen einer Spalte, insbesondere im Fall des Zugriffs auf viele Datensätze.[68]

Erwähnenswert ist die Tatsache, dass eine Datenfragmentierung nicht zwangsläufig zu einer Leistungsverbesserung führt, sondern im Gegenteil zu einer hohen Netzwerkauslastung führen kann. Dies ist dann der Fall, wenn eine Abfrage über mehrere Datenfragmente respektive Knoten ausgeführt wird und letztlich zu einem Ergebnis zusammengeführt werden muss.[69] Beispielhaft hierfür ist das Extrahieren einer kompletten Datenzeile in einer Spaltenfamilien-Datenbank.

[65] Vgl. [EN10, S. 894ff]
[66] Vgl. [Ges+16]
[67] Vgl. [Wie15, S. 245]
[68] Vgl. [Coo+10]
[69] Vgl. [Wie15, S. 246]

3 Methodik und Werkzeuge

Der methodische Teil dieser Arbeit gliedert sich in zwei Blöcke: im ersten werden zwei Vorgehensmodelle zur Leistungsanalyse vorgestellt werden und der zweite widmet sich den Werkzeugen zur Leistungsanalyse. Hierbei wird insbesondere auf das Framework Yahoo! Cloud Serving Benchmark (YCSB) eingegangen, welches aktuell und de facto Standard ist für Leistungsanalysen von Datenbanksystemen.[1]

3.1 Leistungsvergleich (Benchmarking)

Mit dem Begriff Benchmarking wird für gewöhnlich in der Informatik das Ergebnis eines Programms oder eine Abfolge von Programmen (Tests) bezeichnet, welche zum Ziel haben, die Leistung einer Lösung unter bestimmten Bedingungen zu messen und mit anderen Lösungen zu vergleichen.[2] Im Folgenden sollen diese Begrifflichkeit und ihre zugrundeliegenden Konzepte im Zusammenhang mit dem Einsatz von NoSQL-Datenbanksystemen betrachtet werden.

3.1.1 Definition und Aspekte

Im Lichte einer Leistungsanalyse von IT-Systemen definieren Bouckaert et al. [Bou+10] den Terminus *Benchmarking* als Leistungsanalyse als den Prozess des Messens und Auswertens von Ausführungsgeschwindigkeiten, Netzwerkprotokollen, Geräten und Netzwerken mithilfe eines standardisierten Vorgehens unter gleichen Bedingungen. Ziel dieses Benchmarking-Prozesses ist ein angemessener Vergleich zwischen unterschiedlichen Lösungen oder Variationen einer eines Systems in einem Testszenario (SUT: System Under Test).

In diesem Modell besteht ein *Benchmark* aus einer Menge von Spezifikationen, die eine faire Leistungsanalyse eines bestimmten Aspekts eines SUT ermöglichen sollen. Zu diesen vier Spezifikationen[3] respektive Aspekten gehören:

- Das **Testszenario** beinhaltet die ausführliche Versuchsanordnung des Experiments beziehungsweise Tests, die für das Benchmark verwendet werden soll.
- Die **Kriterien**, die zum Vergleich der Ergebnisse der einzelnen Lösungen herangezogen werden.
- Die **Metriken** repräsentieren die quantitative Ausprägung eines Systemkriteriums im Testszenario.
- Der **Benchmarking Score** ist die Summe aller gewichteten Leistungsbewertungen (Score) bezüglich der Metriken der einzelnen Kriterien (siehe Tabelle 3.1).

[1] Vgl. [Kle+15]
[2] Vgl. [Bou+10]
[3] Vgl. [Bou+10]

Testfall	Kriterium	Metrik	Gewichtung	Score
Testfall 1	Kriterium 1	Wert	G_1	...
	Kriterium 2	Wert	G_2	...
	⋮	⋮	⋮	⋮
	Kriterium n	Wert	G_n	...
Testfall 2	Kriterium 1	Wert	G_1	...
	Kriterium 2	Wert	G_2	...
Summe				*Benchmarking Score*

Tabelle 3.1: Ermittlung eines Benchmarking Score

Zu den weiteren Eigenschaften eines Benchmarks gehört die Vergleichbarkeit. Hierunter versteht man, dass es möglich ist, in unabhängig voneinander ausgeführten Tests unterschiedliche Lösungen zu vergleichen. Eine weitere Folge dieser Bedingung ist, dass die Wiederholung eines Benchmarks für eine Lösung zum gleichen Ergebnis führen muss. Diese Anforderungen an ein Benchmark setzen eine wohldefinierte Versuchsanordnung voraus.[4]

Basierend auf dem vorgestellten Modell eines allgemeinen Benchmarks erweitern Barata et al. diese Definition in Bezug auf ein Datenbankmanagementsystem (DBMS):

> „A database benchmark is a standard set of executeable instructions used to measure and compare the relative and quantitative performance of a database management system through the execution of controlled experiments, illustrating the advantages of a system in a given situation."[5]

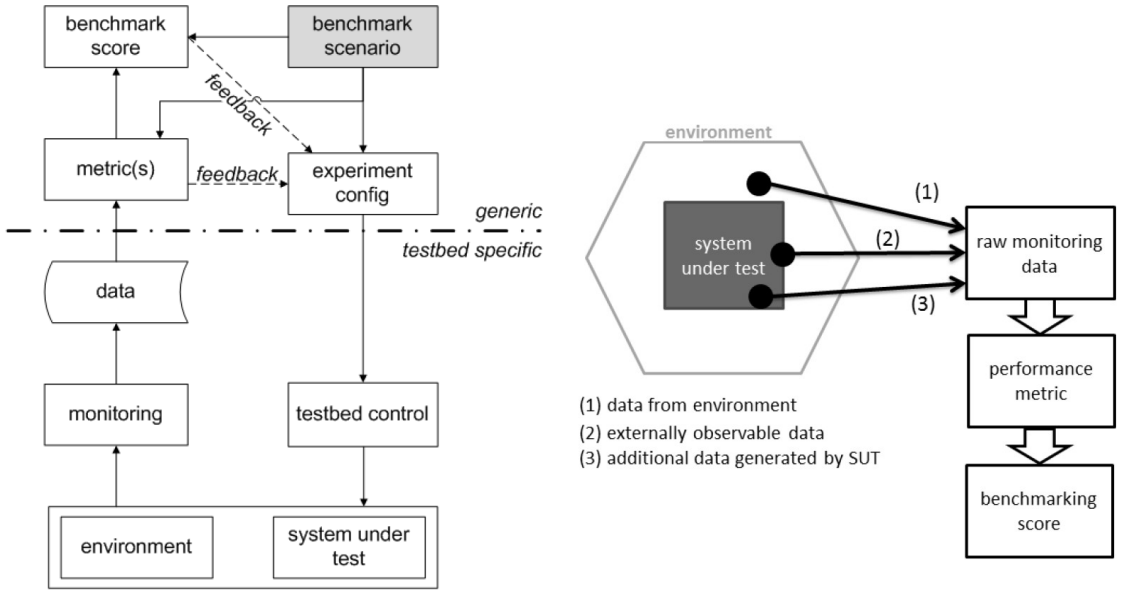
3.1.2 Benchmarking Framework

Die Überblicksdarstellung (siehe Abbildung 3.1(a)[6]) zeigt das *Benchmarking Framework*, welches von Bouckaert et al. [Bou+10] entwickelt wurde und den experimentellen Ansatz dieser Domäne berücksichtigt. Das Rahmenwerk lässt sich auf eine Testumgebung (Testbed) anwenden und besteht aus zwei logischen Teilen:

- Der **generische Teil** umfasst das Testszenario (Benchmark Scenario) und seine Aspekte, welche in Abschnitt 3.1.1 erläutert wurden. Zu beachten ist die experimentelle Konfiguration (Experiment Config[uration]), welche die zu analysierende Lösungsvariante darstellt. Der iterative Ansatz (im Modell dargestellt durch die Feedback-Pfeile) ermöglicht ein inkrementelles Anpassen der Lösungsvarianten mithilfe der im Testszenario erhobenen Daten.[7]

- Der **testumgebungsspezifische** Teil beinhaltet die konkrete Implementierung des Testszenarios. In der logischen Komponente zur Steuerung der Testumgebung (Testbed Control) fließen alle Parameter ein, die die Lösungsvariante im SUT sowie die Testumgebung (Environment) definieren. In der Monitoring-Komponente werden alle Aktivitäten zur Datenerhebung und späterer Bereitstellung (Data) während der Durchführung des Tests gebündelt.[8]

[4] Vgl. [Bou+10]
[5] Zitiert aus [BBF14]
[6] Abbildung entnommen aus [Bou+10]
[7] Vgl. [Bou+10]
[8] Vgl. [Bou+10]



(a) Logischer Aufbau (b) Daten und Metriken

Abbildung 3.1: Das Benchmarking Framework

Der Zusammenhang und die Abhängigkeit von erhobenen Daten und den Metriken des Benchmarks wird in Abbildung 3.1(b)[9] verdeutlicht. Zuerst gilt festzustellen, ob Rohdaten (Raw Data) bezüglich der Messung an folgenden drei Stellen im System erhoben werden können: Aus der Systemumgebung (1: Data from Environment), außerhalb (2: Externally obeservable Data) und innerhalb des SUT (3: Additional Data generated by SUT). Aus diesen Rohdaten werden nun mittels eindeutiger und klarer Definitionen die Metriken (Performance Metric) errechnet. Aus der Summe der gewichteten Einzelbewertungen (Score) errechnet sich der Benchmarking Score (siehe hierzu auch Tabelle 3.1). Auch dieser Schritt setzt eindeutige und klare Definitionen voraus.[10]

3.1.3 Cloud Service Evaluation

Li et al. betrachten in ihrem Verfahren zur experimentellen Bewertung von Diensten im Cloud Computing (Methodology For Cloud Service Evaluation) NoSQL-Datenbanksysteme als Dienst im Cloud Computing (Cloud Service). Die Autoren haben auf Grundlage einer umfassenden und strukturellen Untersuchung zu Cloud-Service-Anforderungen und bestehenden Benchmarks folgende Kategorien von Anwendungsfällen definiert.[11]

- **Cloud Resource Exploration:** Hierunter fallen alle Anwendungsfälle, die der Erforschung des Dienstes und seiner verfügbaren Ressourcen (Beispiel: CPU) dienen.
- **Business Computing:** Die Nutzung von Cloud Services zum technischen Betrieb von Geschäftsanwendungen (Beispiel: E-Shop).
- **Scientific Computing:** Die wissenschaftliche Nutzung von Diensten im Cloud Computing (Beispiel: Vergleich der Verarbeitungsgeschwindigkeit zwischen unterschiedlichen Algorithmen).

[9] Abbildung entnommen und modifiziert aus [Bou+10]
[10] Vgl. [Bou+10]
[11] Vgl. [LOR16]

Physikalische Eigenschaften	Kapazitätseigenschaften	
	primär	sekundär
Netzwerk/Kommunikation	Latenz (Zeit)	Elastizität
Verarbeitung: CPU	Verarbeitungsgeschwindigkeit	Skalierbarkeit
Volatiler Speicher: RAM	Datendurchsatz	Verfügbarkeit
Persistenter Speicher: SSD/HDD	Dauerhaftigkeit	Zuverlässigkeit

Tabelle 3.2: Leistungseigenschaften eines Cloud Service

Im Modell der Cloud Service Evaluation werden drei Eigenschaften eines Dienstes (Cloud Service Feature) untersucht: Leistung (Performance), Wirtschaftlichkeit (Economics) und Sicherheit (Security). Im Folgenden soll ausschließlich die Leistung in unterschiedlichen Aspekten betrachtet werden. Li et al. führen an, dass bei einer experimentell untersuchten Leistungseigenschaft (Evaluated Performance Feature) typischerweise physikalischen Einheiten kapazitativen zugeordnet werden.[12] Zu den physikalischen Einheiten zählt das Netzwerk bestehend aus Knoten, Hardware, Serverinstanzen sowie der Kommunikation zwischen ihnen und den Clients. Bei einer Analyse wird nun die primäre (unmittelbare) oder sekundäre (mittelbare) Kapazität einer physikalischen Eigenschaft ermittelt. Die Tabelle 3.2 ordnet die Systemqualitäten und Klassifikationen aus den Abschnitten 2.2 und 2.3 in dieses Modell sowie mögliche Zuordnungen ein.

3.1.4 Cloud Evaluation Experiment Methodology

Die Autoren der Cloud Service Evaluation führen das evidenzbasierte Verfahren *Cloud Evaluation Experiment Methodology (CEEM)* ein, das auf der Evaluation klassischer Computersysteme basiert – zu denen sie Cloud Computing als Implementierung eines Paradigmas zählen – und zu bereits verfügbaren Untersuchungen zu Cloud Systemen. Ferner finden betreffend des Versuchsaufbaus und der statistischen Analyse in dieser Methode die Richtlinien zum Design of Experiments (DOE) Anwendung. Das CEEM-Modell stellt sich als Prozess dar, wie in der Abbildung 3.2[13] dargestellt, welcher aus zwei logischen Teilen besteht. Der erste Teil beinhaltet ein sequenzielles Ablaufen von Aktivitäten vor der Ausführung des Experiments sowie die Dokumentation dessen als letzte Handlung des Gesamtprozess (Schritte 1–6 und 10). Der zweite Teil umfasst eine iterative Sequenz von Aktivitäten während der Durchführung des Experiments und ermöglicht eine inkrementelle Modifikation des Versuchsaufbaus und die Wiederholung des Experiments (Schritte 7–9).[14]

Insgesamt sind zur Durchführung der Evaluation eines Cloud Service nach CEEM die nachfolgenden zehn Schritte notwendig:[15]

1. **Identifizierung der Anforderungen (Requirement Recognition):** Eine genaue und eindeutige Formulierung der Anforderungen ermöglicht erst ein klares Verständnis für das Ziel und den Zweck der Untersuchung (Experiment).

2. **Identifizierung der Diensteigenschaften (Service Feature Identification):** Aus den ermittelten Anforderungen werden die zu untersuchenden Diensteigenschaften aus Leistung, Wirtschaftlichkeit und Sicherheit identifiziert.

[12] Vgl. [LOR16]

[13] Abbildung entnommen und modifiziert von `https://cecs.anu.edu.au/files/posters/2014/u4938849.pdf` (besucht am 18.01.2017)

[14] Vgl. [LOZ13]

[15] Vgl. [LOR16], [LOZ13]

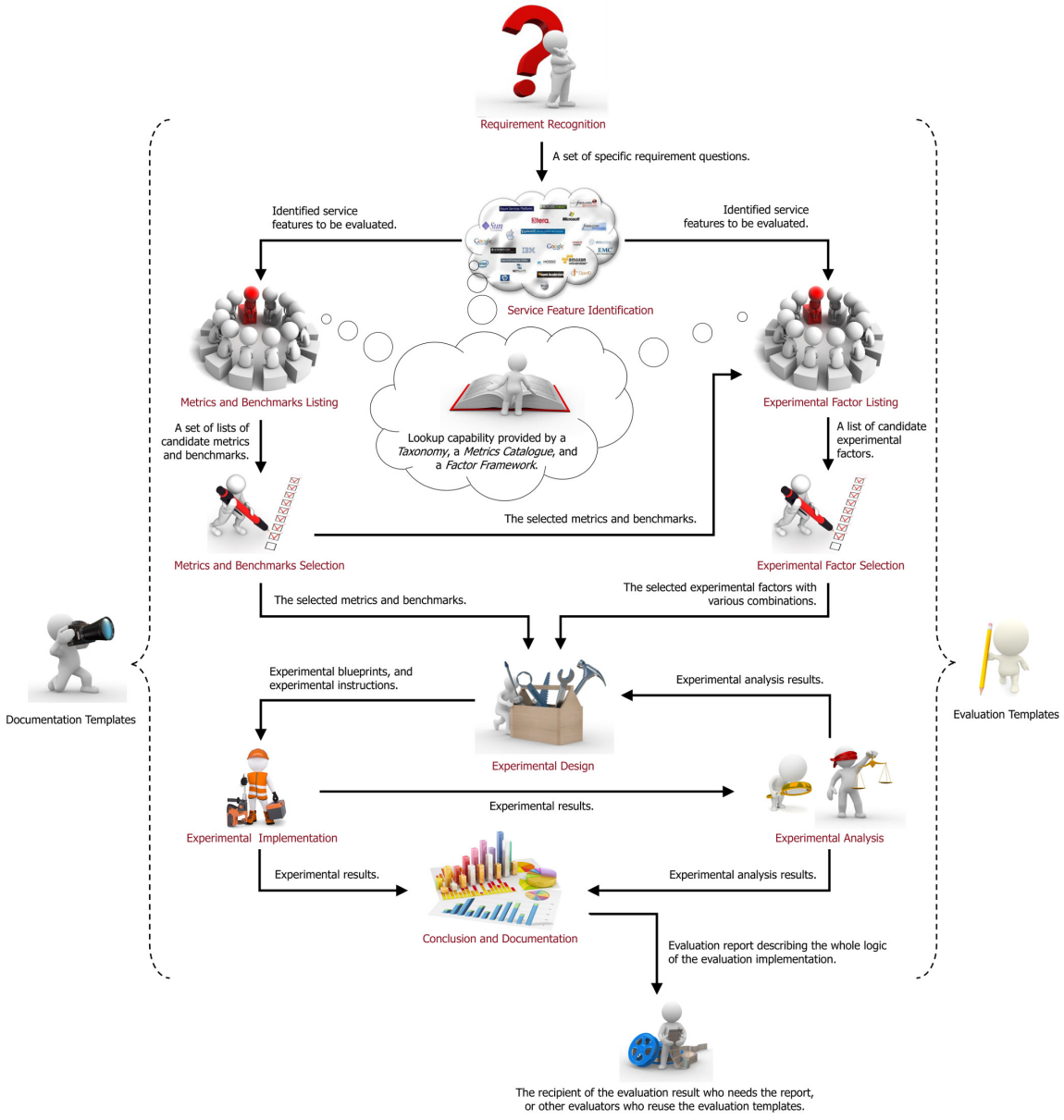


Abbildung 3.2: Cloud Evaluation Experiment Methodology (CEEM)

3. **Verzeichnis der Metriken und Vergleichsgrößen (Listing of Metrics and Benchmark Listing):** Alle Metriken und Vergleichsgrößen bezüglich der zu untersuchenden Eigenschaften des Dienstes sollen ermittelt und verzeichnet werden, um alle Abhängigkeiten und Zusammenhänge zwischen ihnen aufzuzeigen.

4. **Auswahl der Metriken und Vergleichsgrößen (Selection of Metrics and Benchmarks):** Die Auswahl an Metriken beeinflusst unmittelbar die Auswahl der zur Verfügung stehenden Vergleichsgrößen für das Benchmark und stellt einen wichtigen Schritt der Cloud Service Evaluation dar.

5. **Verzeichnis der Parameter und Variablen im Experiment (Listing of Experimental Factors):** Die Faktoren, also alle Parameter und Variablen, welche die Diensteigenschaften beeinflussen können, sollen aufgelistet werden. Dieses Verzeichnis bildet die Grundlage für die Diskussion und Auswahl der zu evaluierenden Parameter und Variablen im Experiment.

6. **Auswahl der Parameter und Variablen im Experiment (Selection of Experimental Factors):** Der letzte Schritt vor der Durchführung liegt in der Auswahl der zuvor evaluierten Parameter und Variablen für das Experiment. Diese Auswahl sollte auch zum besseren Verständnis mittelbare (indirekte) Parameter und Variablen des Untersuchungsgegenstandes beinhalten.

7. **Versuchsanordnung (Experimental Design):** Die ausgewählten Parameter werden nun physikalischen Eigenschaften im System zugeordnet, so dass zu den ausgewählten Variablen Analysedaten erhoben werden können. In einem iterativen Prozess beginnt hier die Durchführung mit modifizierten Faktoren des Experiments anhand zuvor erfolgter Versuchsdurchführung und -analyse.

8. **Versuchsdurchführung (Experimental Implementation):** Die Versuchsdurchführung erstreckt sich von der konkreten Konfiguration des zu untersuchenden Systems bis zur tatsächlichen Durchführung des Versuchs mit dem Ziel der Datenerhebung. Es ist sicherzustellen, dass die Versuchsdurchführung der intendierten Versuchsanordnung entspricht.

9. **Versuchsanalyse (Experimental Analysis):** Für die Analyse des Versuchs anhand der erhobenen Daten sind statistische Auswertungsverfahren zu verwenden. Die Datenanalyse und Interpretation der Ergebnisse kann durch Visualisierungen verdeutlicht werden.

10. **Ergebnis und Dokumentation (Conclusion and Documentation):** Die Dokumentation der Evaluation enthält die Beschreibung der Metriken, Vergleichsgrößen, der Parameter und Variablen sowie die aufbereiten Ergebnisse aus der Versuchsanalyse.

3.2 Yahoo! Cloud Serving Benchmark (YCSB)

Das *Yahoo! Cloud Serving Benchmark (YCSB)* wurde im Jahr 2010 von Cooper et al. [Coob] vorgestellt und wird seitdem fortlaufend weiterentwickelt. Mit diesem Framework sollen die vielen neuartigen NoSQL-Datenbanken standardisiert untersuchbar sein durch künstlich erzeugte Arbeitslast (siehe hierzu Abschnitt 4.1.2). Dabei werden die Funktionen und das Leistungsvermögen experimentell erforscht.[16]

[16] Vgl. [Cooa]

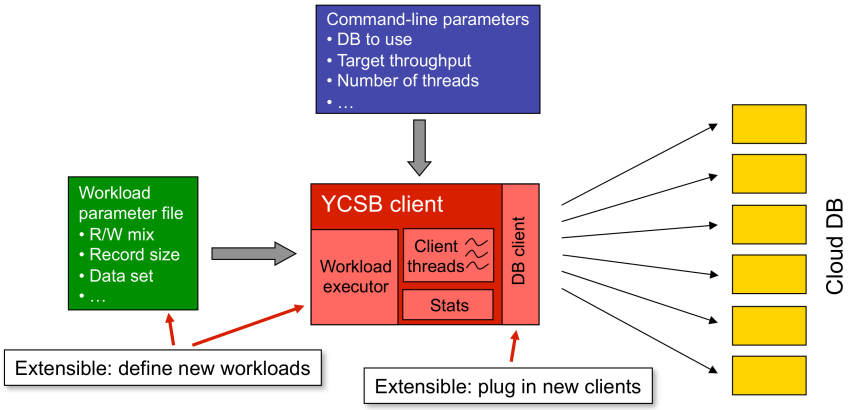


Abbildung 3.3: Module des Yahoo! Cloud Service Frameworks (YCSB)

3.2.1 Architektur und Schnittstellen

Die Abbildung 3.3[17] zeigt den modularen Aufbau des YCSB-Frameworks. Dieser besteht aus dem *Workload Generator*, der Datei- oder Kommandozeilen-basiert ist, und an diese Merkmale zur Generierung der Arbeitslast übergeben werden. Mit dieser individuellen Arbeitslast werden die einzelnen Produkte, Lösungen oder Variationen getestet. Außerdem enthält das Framework das Modul *YCSB Client* (ebenfalls Kommandozeilen-basiert), welches aus den Komponenten *Workload Executor* (führt die Daten-getriebene Arbeitslast aus) und den *Client Threads*, die die Arbeitslast um den Prozess-getriebenen Anteil vervollständigen.[18]

Bemerkenswert ist das Modul *DB Client*, welches die unterschiedlichen Datenoperationen und Zugriffsmechanismen auf eine NoSQL-Datenbank (über eine API, HTTP/REST, JNI) kapselt und auf die im Framework standardisierten Datenoperationen (`read`, `insert`, `update`, `delete`, `scan`) abbildet. Es ist insofern erweiterbar, als es für jede Datenbank implementiert werden muss, falls hierfür noch keine Implementierung existiert. Die beim Ausführen der Arbeitslast erhobenen Daten werden durch das *Statistik-Modul (Stats)* protokolliert und können so später weitergehend ausgewertet werden.[19]

Der YCSB-Client im Framework ist auf Erweiterbarkeit ausgelegt und gliedert sich modular in mehrere logische Schichten (Tier).

- **Tier 1 – Leistung:** Innerhalb dieser Schicht werden alle Merkmale zur Leistungsmessung der Latenz und des Datendurchsatzes gebündelt. Als Ergebnis fungieren hier erhobene Rohdaten, die interpretierbar sind bezüglich der gestellten Anforderungen.
- **Tier 2 – Skalierung:** Diese logische Einheit umfasst alle Metriken zum Verhalten des zu testenden Systems in Bezug auf vertikale und elastische horizontale Skalierung.
- **Tier 3 – Verfügbarkeit:** Diese Schicht ist für eine zukünftige Erweiterung um Messungen hinsichtlich der Verfügbarkeit vorgesehen.
- **Tier 4 – Replikation:** In diesem Modul sollen alle Tests zum Replikationsverhalten gesammelt werden, um die das Framework erweitert werden soll.[20]

[17] Abbildung entnommen aus [Cooa]
[18] Vgl. [Coo+10], [Cooa], [Pat+11]
[19] Vgl. [Coob]
[20] Vgl. [Coo+10]

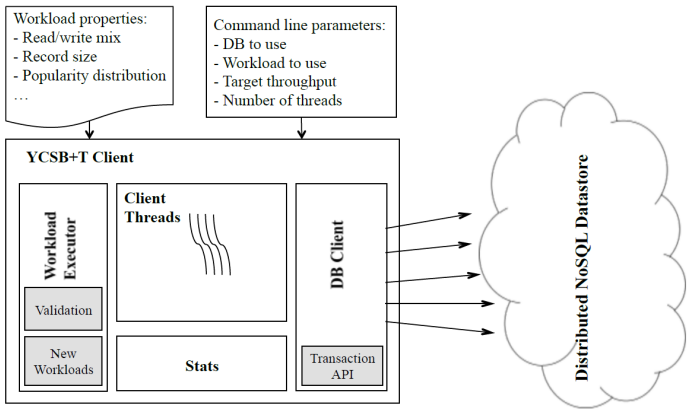


Abbildung 3.4: Transaktionale Erweiterung des YCSB-Frameworks (YCSB+T)

- **Tier 5 – Transaktionaler Mehraufwand:** Dieses Modul misst den bedingten Mehraufwand beim Ausführen mehrerer Datenoperationen innerhalb einer Transaktion.
- **Tier 6 – Datenkonsistenz:** Innerhalb dieser Schicht werden alle Methoden zur Evaluierung der Datenkonsistenz gebündelt.[21]

3.2.2 Erweiterungen

Dey et al. [Dey+14] nutzen die konzeptionelle Möglichkeit zur Erweiterung des YCSB-Frameworks um den Aspekt der Transaktionalität in homogenen und verteilten Schlüssel-Wert-Datenbanken. Um das Verhalten hinsichtlich Leistung, Skalierbarkeit, transaktionalem Mehraufwand und Datenkonsistenz zu untersuchen, fügen sie den YCSB-Modulen Workload Executor und DB Client weitere Funktionalitäten hinzu, wie in Abbildung 3.4[22] dargestellt. Der Workload Executor wird dabei um Logik bezüglich Transaktionen erweitert. In diesem Fall handelt es sich dabei um Mechanismen zur Simulation einer Banküberweisung. Ferner erfordert die Transaktionsbedingung eine Validation, welche dem Workload Executor als weitere Kompenente hinzugefügt wurde. Die technische Umsetzung der Transaktion geschieht auf Datenbankebene in der Komponente DB Client mittels eines transaktionalen Zugriffsprotokolls.[23]

Ein Beispiel für eine mächtige Erweiterung des YCSB-Frameworks stellen Patil et al. [Pat+11] vor. Der Unterschied zur Basisversion ist in Abbildung 3.5[24] ersichtlich, wo die neuen Komponenten der Erweiterung mit schwarzem Hintergrund dargestellt sind. Diese Erweiterungen[25] setzen sich wie folgt zusammen:

- Für die erweiterten Funktionalitäten des Rahmenwerks sind weitere Pakete zur Generierung der Arbeitslast („New workloads“) hinzugefügt worden. Diese lassen sich über neue Parameter („Extensions“) konfigurieren und steuern. Erwähnenswert ist hierbei die Möglichkeit zur Definition von Wertebereichen für Datenbanksysteme, welche die Tabellen anhand bestimmter Schlüssel oder Größen über mehrere Knoten verteilen.

[21] Vgl. [Dey+14]
[22] Abbildung entnommen aus [Dey+14]
[23] Vgl. [Dey+14]
[24] Abbildung entnommen aus [Pat+11]
[25] Vgl. [Pat+11]

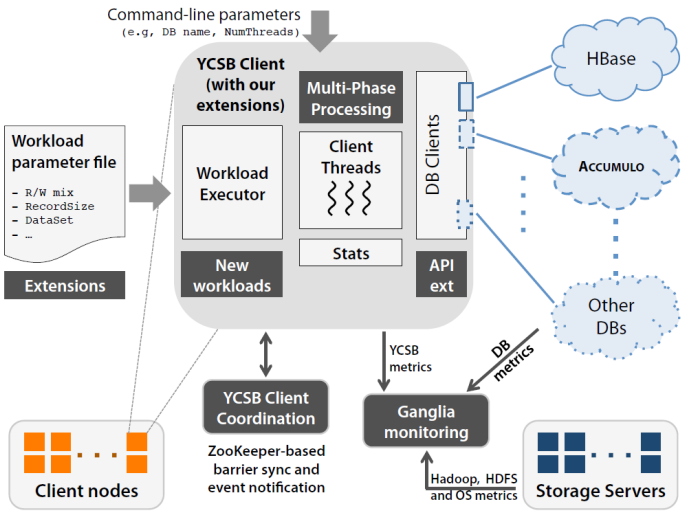


Abbildung 3.5: Erweiterte Architektur des YCSB-Frameworks (YCSB++)

- Ein weiteres Modul ermöglicht das parallele Testen über mehrere Serverinstanzen („Client nodes“) mit über einhundert Threads, welche zentral über die Komponente „YCSB Client Coordination“ gesteuert werden.
- Die Evaluierung der Datenkonsistenz wird ebenfalls über das Modul realisiert, welches zum parallelen Testen verwendet wird. Es ändert einen Datensatz mit bekannten Werten. Anschließend misst es die Latenz bei der Replikation der Daten durch direktes asynchrones Abfragen auf den zu testenden Knoten respektive Serverinstanzen und letztlich einem Wertevergleich des gesuchten Datensatzes.
- Eine Erweiterung der Schnittstelle für den Zugriff auf Datenbanken („API ext“) ermöglicht das native Generieren und Importieren massiver Datensätze für bestimmte Datenbanktypen in proprietären Dateisystemen.
- Ein weiteres Modul realisiert das „Server-side Filtering“. Darunter wird das Verlagern der Datenverarbeitung vom Client zum Server bezeichnet. Dies hat zum Ziel jene die Datenmenge zu reduzieren, welche vom Datenträger gelesen und über das Netzwerk übertragen werden muss. Die Unterstützung dieser Funktion ist abhängig vom Datenbanktyp und wird im Zweifelsfall von der Komponente clientseitig umgesetzt („Multi-Phase Processing“).
- Für Datenbanktypen, die eine Zugriffskontrolle auf Basis von Tabellen, Entitäten, Spalten, Spaltenfamilien, Zeilen oder Tupel (Zellen oder Schlüssel-Wert) anbieten, lassen sich mit dem Modul „Multi-Phase Processing“ Leistungsmessungen umsetzen, um den Einfluss des Zugriffsschutzes auf das Leistungsverhalten zu analysieren.
- Eine system- und clusterweite Überwachung und Darstellung der CPU-Auslastung, des Verbrauches von Arbeits- und Festplattenspeicher sowie des Netzwerkverkehrs ermöglicht die Komponente „Ganglia monitoring“, welche in regelmäßigen Abständen diese Metriken erhebt und in Zusammenhang mit der ablaufenden Leistungsmessung stellt.

3.2.3 Vergleich zu anderen Benchmarks

Das Transaction Processing Performance Council (TPC) ist ein Non-Profit-Konsortium verschiedener IT-Unternehmen, welches zum Ziel hat, mittels standardisierter Benchmarks Aussagen zur Leistungsfähigkeit relationaler Datenbankmanagementsysteme zu ermöglichen. Die vier aktuellen TPC-Benchmarks basieren jeweils auf der Simulation eines Geschäftsvorfalls.[26] Im Folgenden soll das TPC-H Benchmark [Couc] betrachtet werden, welches ein Entscheidungsunterstützungssystem (DSS: Decision Support System) simuliert. Bei einem DSS handelt es sich um ein datenintensives Informationssystem, aus dem die zur Entscheidungsunterstützung relevanten Informationen ermittelt und aufbereitet werden. Zu beachten sind hier die besonderen Anforderungen bei großen Datenmengen hinsichtlich komplexer Abfragen sowie konkurrierender Schreibzugriffe und Transaktionen.[27]

YCSB versteht sich als Rahmenwerk und Werkzeug für Datenbanken und nicht als Standard. Trotz des unterschiedlichen Ansatzes gegenüber TPC lassen sich doch beide Benchmarks anhand mehrere Aspekte miteinander vergleichen:

- **Anwendbarkeit:** Das TPC-H Benchmark sollte auf jeder relationalen Datenbank ausführbar sein, die die gängigen Datentypen und Transaktionalität unterstützt. Ähnlich hierzu verhält sich das YCSB-Rahmenwerk, welches unterschiedliche Datenmodelle und Datenbanktypen unterstützt.

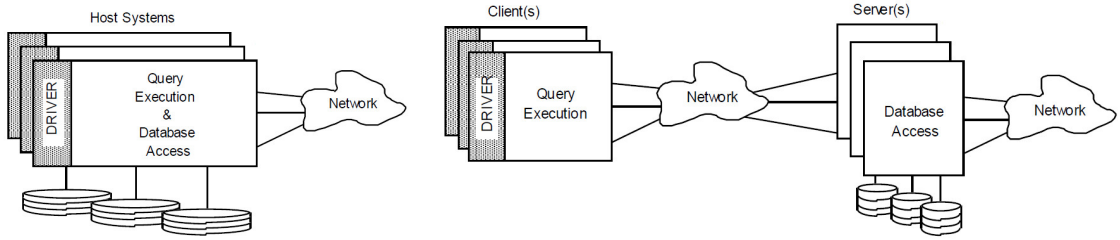
- **Komponenten:** Das zur Verfügung gestellte Softwarepaket für den TPC-H Benchmark besteht aus den zwei wesentlichen Komponenten. Zum einen verfügt es über einen Datengenerator zur Erzeugung und Import des skalierbaren Datenvolumens sowie der Komponente zum Ausführen von Abfragen und zum Erheben der Metriken. Ähnlich wie das YCSB-Framework verfügt diese Kernkomponente über verschiedene Schnittstellen zum Bedienen der unterschiedlichen Datenbankimplementierungen.

- **Arbeitslast:** Der Datengenerator stellt eine Vergleichbarkeit der Datensätze sicher durch gleiche Feldlängen und festgelegte Wahrscheinlichkeiten bezüglich der Auswahl eines Datensatzes für eine Datenoperation. Dieses Vorgehen unterscheidet sich zum YCSB-Framework hauptsächlich in der Auswahl der Datenoperationen, die beim TPC-H Benchmark vor dem Test bereits festgelegt sind und nicht zufällig anhand von Wahrscheinlichkeiten erfolgen.

- **Architektur:** Die Abbildung 3.6[28] zeigt die zwei architektonischen Varianten, mit dem das TPC-H Benchmark ausgeführt werden kann. In der Variante *Host-System* (Abbildung 3.6(a)) erfolgen der konkrete Datenzugriff und die Abfragebearbeitung auf demselben physikalischen System. Dabei können mehrere Hosts über ein Netzwerk miteinander zu einem System gekoppelt werden. Im Unterschied hierzu erfolgt im *Client-Server-System* (Abbildung 3.6(b)) die Datenermittlung und Abfragebearbeitung auf unterschiedlichen physikalischen Ressourcen, die ebenfalls über ein Netzwerk verbunden und vertikal skalierbar sind.

- **Durchführung:** Der TPC-H Benchmark ist definiert als Ausführung eines Last- und Geschwindigkeitstests. Der Lasttest beinhaltet das Erstellen der Datentabellen und das Importieren des generierten Datensatzes. Der Geschwindigkeitstest umfasst 22

[26] Vgl. `http://www.tpc.org/information/about/abouttpc.asp` (besucht am 09.04.2017)
[27] Vgl. [BBF14]
[28] Abbildungen entnommen aus [Couc, S. 101]



(a) Host-System (b) Client-Server-System

Abbildung 3.6: Architektonische Varianten des TPC-H Benchmark

nur lesende Abfragen sowie zwei Update-Anweisungen und wird zwei Mal hintereinander ausgeführt.[29]

- **Metriken:** Das System aus Metriken, welche im TPC-H Benchmark verwendet werden, basiert auf dem Parameter des Skalierungsfaktor. In der Regel ist das die zu verarbeitenden Datenmenge oder die Anzahl simultaner Threads. Mit den Daten aus den Messungen werden hieraus numerischen Größen errechnet, wie die Verarbeitungsgeschwindigkeit und Durchsatz. Diese Werte sind wiederum selbst Bestandteil der Berechnung weiterer primärer und sekundärer Metriken, aus dem sich der TPC-H Benchmark bildet.[30]

- **Transaktionen:** Die Benchmarks TPC-C [Coua] und TPC-E [Coub] simulieren eine realistische Transaktionslast aus verschiedenen Transaktionstypen und ihren Interdependenzen, die weitere Transaktionen anstoßen können. Diese Benchmarks dienten als konzeptionelle Vorlage für die Erweiterungen des YCSB-Frameworks hinsichtlich transaktionaler Eigenschaften (YCSB+T).[31]

[29] Vgl. [Couc, S. 92]
[30] Vgl. [Couc, S. 98ff]
[31] Vgl. [Dey+14]

4 Leistungsmessung

Im folgenden Kapitel werden die verschiedenen Dimensionen und Aspekte der Leistungsmessung von NoSQL-Datenbanken beschrieben. Die Anforderungen, die nicht-relationale Datenbanksysteme erfüllen sollen, und ihre Zugehörigkeit zu den Cloud-Systemen[1] bedingen eine Unterscheidung in leistungs-, betriebs- und sicherheitsbezogenen Dimensionen und Aspekten.[2] Im Folgenden sollen nur die leistungsrelevanten Aspekte im Sinne der in Abschnitt 2.1.2 vorgestellten Anforderungen betrachtet werden.

4.1 Parameter und Variablen

Eine Leistungsmessung kann als Test nicht-funktionaler Aspekte verstanden werden, welche sich im Wesentlichen in Geschwindigkeits-, Last-, Skalierungs- und Verfügbarkeitstests gliedern lassen.[3] Ebenso lassen sich über eine Leistungsmessung funktionale Aspekte einer Datenbank betrachten, wie unterstützte Datenmodelle, Abfragesprachen und Schnittstellen, Konsistenz sowie weitere Werkzeuge zur Wartung des Datenbanksystems.[4] Die folgenden Abschnitte konkretisieren die allgemeinen Parameter und Faktoren eines Cloud-Dienstes bezogen auf die Anforderungen an ein NoSQL-Datenbanksystem.

4.1.1 Datenoperationen

Die funktionale Basis zur Messung, Analyse und Bewertung von Datenbanksystemen werden gebildet vier fundamentalen Datenoperationen, deren Bezeichnungen in Abhängigkeit zum Datenbanktyp stehen. Bei relationalen Datenbanken spricht man hierbei von CRUD-Operationen und meint das Erzeugen (Create), Lesen (Read), Aktualisieren (Update) und Entfernen (Delete) von Datensätzen. Bei nicht-relationalen Datenbanken existieren hierfür keine einheitlichen Benennungen aufgrund der Arbeitsweisen der unterschiedlichen Datenbanktypen.

Für Schlüssel-Wert-Datenbanken können die folgenden vier fundamentalen Datenoperationen auf Basis der verwendeten Hashtabelle zum Lesen und Schreiben eines Datensatzes definiert werden:[5]

- **Instanziierung (Instantiate)** eines Bucket zur Speicherbelegung.
- **Schreiben (Write)** umfasst das Suchen nach einem Schlüssel-Wert-Paar. Falls nicht gefunden, schreiben ansonsten überschreiben dieses Paares, falls er nicht gefunden werden kann, respektive das Überschreiben des gefunden Wertes für einen gesuchten Schlüssel. Dies entspricht den Operationen Erzeugen und Aktualisieren im bekannten CRUD-Modell.
- **Lesen (Read)** und **Entfernen (Delete)** entsprechen den gleichnamigen Operationen im CRUD-Modell.

[1] Siehe hierzu Abschnitt 2.2.1
[2] Vgl. [Iye16]
[3] Vgl. [DDS16]
[4] Vgl. [LM13]
[5] Vgl. [LM13]

Arbeitslast	Anwendungsfall
50% Read - 50% Write	Sessionspeicher
100% Read	Rechenprozesse
100% Write	Logging
100% Read-Modify-Write	Aktualisierung
100% Scan	Suchen

Tabelle 4.1: Arbeitslast für typische Anwendungen

Die Auflistung der vier elementaren Datenoperationen von Cooper et al. [Coo+10] verhält sich generisch zur Datenbankimplementierung und lässt sich somit für alle nichtrelationalen Systeme anwenden. Sie umfasst Einfügen (Insert), Aktualisieren (Update), Lesen (Read) und Durchsuchen (Scan) eines Objektes (Schlüssel-Wert-Paar, Spalte, Dokument oder Graph) und eignet sich eher für den Vergleich zwischen unterschiedlichen Datenbanktypen und Datenmodellen.[6]

4.1.2 Arbeitslast (Workload)

Im Allgemeinen bezeichnet man bei Datenbanken mit *Workload* die gesamte *Arbeitslast*, die durch alle Anfragen der Benutzer und Anwendungen eines Datenbanksystems (innerhalb eines bestimmten Zeitraumes) aufkommt.[7] Im Zusammenhang mit der Leistungsanalyse bei Datenbanksystemen versteht man unter *Arbeitslast* im Speziellen die Gesamtheit aller Datenoperationen für eine Leistungsmessung. Im weiteren Sinn kann unter der Begrifflichkeit auch die die Datenoperationen, die auf der Datenmenge ausgeführt werden.[8] Eine Vergleichbarkeit zwischen den getesteten Systemen wird dadurch ermöglicht, dass die Bedingung der Arbeitslast für jede untersuchte Umgebung innerhalb eines Experiments gleich ist.

Die Leistungsmessung wird für gewöhnlich mit einer Menge zusammengehöriger Arbeitslast durchgeführt. Diese bestehen aus jeweils aus einem wohldefinierten Verhältnis von Schreib- und Leseoperationen, gleich großen Datenmengen für ein Datenfeld und unterschiedlich großen Datenmengen pro Arbeitslast sowie weiteren relevanten Parametern bestehen, welche Aufschluss über das Leistungsverhalten geben sollen.[9] Die Tabelle 4.1 zeigt exemplarisch die Zusammensetzung der Arbeitslast für bekannte Anwendungsfälle.

Die Auswahl der Datenoperationen für eine Arbeitslast sollte sich am Anwendungsfall orientieren, für den die Leistungsanalyse und der Leistungsvergleich durchgeführt wird. Im Kontext von Big Data und dem YCSB-Framework lassen sich grob fünf Anwendungsfälle unterscheiden, wie sie in Tabelle 4.1 dargestellt sind.[10] Es lässt sich hierbei unterscheiden zwischen dem Ansatz der möglichst anwendungsnahen Arbeitslast für konkrete Anwendungsfälle und einer breit gestreuten Arbeitslast zur Erforschung des Leistungsverhalten in seinen Extremen. Die Zusammensetzung der Datenoperationen für eine Arbeitslast erfolgt anhand einer multinominalen Verteilung.[11]

[6] Vgl. [Coo+10]
[7] Vgl. [Koo]
[8] Vgl. [Coo+10]
[9] Vgl. [Kle+15], [LM13] u.a.
[10] Vgl. [Coo+10], [SE16]
[11] Vgl. [Coo+10]

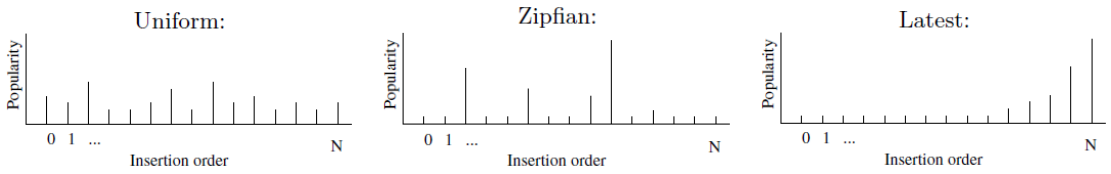


Abbildung 4.1: Mögliche Verteilungen im Workload

Ferner ist für die Arbeitslast die Auswahl jener Datensätze relevant, für welche die Datenoperationen erfolgen sollen. Im generischen Ansatz von Cooper et al. [Coo+10] werden hierzu drei Verfahren zur Verteilung der auszuwählenden Datensätze vorgestellt:

- Bei einer **Gleichverteilung (Uniform)** ist die Wahrscheinlichkeit für jeden Datensatz annähernd gleich groß.
- Bei der **Zipf-Verteilung (Zipfian)** haben bestimmte Datensätze eine deutlich höhere Wahrscheinlichkeit als bestimmte andere Datensätze.
- Gleiches gilt bei der **Verteilung nach dem letzten Rang (Latest)**, jedoch ist hier im Unterschied zur Zipf-Verteilung die Wahrscheinlichkeit höher mit zunehmendem Rang.

Die Abbildung 4.1[12] zeigt die drei Arten der Verteilung. Auf der Abszissenachse sind die Datensätze bezüglich des Ranges in Schreibreihenfolge geordnet. Die Ordinaten beschreibt mittels einer Wahrscheinlichkeitsdichtefunktion die Häufigkeit mit der die Datensätze für eine Datenoperation ausgewählt werden.

Der anwendungsbezogenen Definition von *Arbeitslast* von Ameri et al. [Ame+16] liegt die Annahme zugrunde, dass ein System in Bezug zur konkreten Anwendung steht und deswegen variiert. Es wird unterschieden zwischen einer vektorbasierten Leistungsanalyse, bei der die Leistung eines Datenbanksystems als eine bestimmte Anzahl von Vektoren betrachtet wird (siehe *primäre und sekundäre Leistungseigenschaften* in Tabelle 3.2), und einer sogenannten nachstellenden Leistungsanalyse (trace-based Benchmark). Letztere beruht auf der Idee, dass die Leistung einer Datenbank abhängig ist von dedizierten, also anwendungsspezifischen Anfragen, die zu bewältigen sind.[13] Bei dieser Art von Leistungsanalyse wird für die Messung der reale Anwendungsfall nachgestellt.

Dieses Verständnis von Arbeitslast ist geprägt von den Anforderungen an riesige Datenmengen (umfangreicher Datenbestand, Vielfalt von Datenformaten und hohe Verarbeitungsgeschwindigkeit) und unterscheidet sich von Cooper et al. [Coo+10] in folgenden Punkten:

- Die Arbeitslast bezieht sich hierbei auf einen konkreten Vergleich zwischen explizit unterschiedlichen Lösungen bezüglich des Datenmodells und ist geprägt von sehr spezifischen und komplexen Datenstrukturen und Abfragen.
- Dies führt dazu, dass für jede Lösung individuelle Arbeitslast („synthetische oder künstliche Datenmenge“) erzeugt werden, die über einen gleichfalls zu definierenden Zeitabschnitt abgefragt werden. Datenoperationen und Zeitintervalle lassen sich über eine Standardabweichung der Gaußschen Normalverteilung variieren.
- Die Arbeitslast für eine zu untersuchende Lösung kann sich aufgrund des Datenbanktyps und Datenmodells in der Datenstruktur und der verwendeten Datentypen unterscheiden.

[12] Abbildung entnommen aus [Coo+10]
[13] Vgl. [Ame+16]

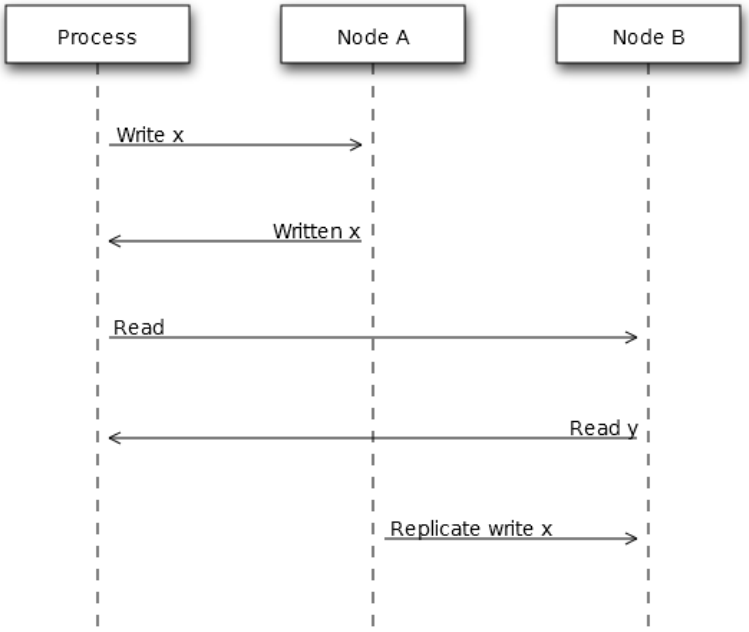


Abbildung 4.2: Lesen veralteter Daten (Stale Read)

- Mit dieser Definition lassen sich auch die Ergebnisse einer Leistungsmessung zwischen relationalen und nicht-relationalen Datenbanksystemen vergleichen. Trotz der Tatsache, dass SQL eine deklarative Abfragesprache ist und die meisten NoSQL-Abfragesprachen prozedural sind.[14]

4.1.3 Datenkonsistenz und Quoren

In den Abschnitten 2.1.3 und 2.1.4 wurde gezeigt, dass sich das Modell bezüglich der Datenkonsistenz bei verteilten Datenbanksystemen von dem traditioneller RDBMS ohne Replikation unterscheidet. Nicht-relationale Datenbanksysteme werden typischerweise als eine verteilte Shared-Nothing-Architektur betrieben, bei der jeder Knoten unabhängig und eigenständig seine Aufgaben mit seinem eigenen Prozessor und dem zugeordneten Speicher erfüllt. Dieses Parallelisierungskonzept bedingt aber aufgrund der Netzwerklatenz, dass Fehler aufgrund der Verfügbarkeit auftreten können, es zum Lesen veralteter Daten (Stale Read) kommt und auch Aktualisierungen (Lost Update) verloren gehen.[15] Zum *Stale Read* kommt es in einem Datenbanksystem mit Replikation, wenn von einem Knoten gelesen wird, der nicht dem aktuellen Datenstand entspricht, wie im Sequenzdiagramm in Abbildung 4.2 dargestellt. Ein *Lost Update* bezeichnet hingegen den Vorgang, wenn eine Änderung nicht an alle Replikationen weitergegeben wird.

Für NoSQL-Datenbanksysteme lässt sich dennoch ein Konzept zur starken Datenkonsistenz implementieren, bei der alle Schreib- und Lesezugriffe auf den Knoten so synchronisiert werden, dass die beschriebenen Phänomene nicht eintreten. Dies erfordert einen hohen Ressourcenaufwand. Das Konzept der *sequenziellen Konsistenz (Sequential Consistency)* stammt aus der Parallelen Programmierung und ordnet die Schreib- und Lesezugriffe

[14] Vgl. [Ame+16]
[15] Vgl. [Wie15, S. 295]

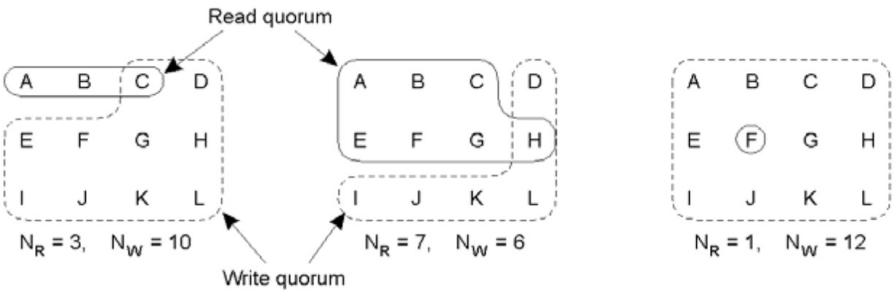


Abbildung 4.3: Quoren

auf allen Knoten. Es stellt somit sicher, dass die Schreibzugriffe auf den Knoten in der gleichen Reihenfolge ausgeführt werden. Dieses Vorgehen schließt Transaktionen aus.[16]

Einen flexiblen Mechanismus zur Vermeidung des Lesens veralteter Daten und verloren gegangener Aktualisierungen sind Quoren für Schreib- und Lesezugriffe. Ein *Quorum für einen Lesezugriff (Read Quorum)* ist definiert als Teilmenge aller Knoten, die für eine erfolgreiche Leseoperation dasselbe Ergebnis zurückgeben müssen. Analog gilt die Definition für das *Quorum für einen Schreibzugriff (Write Quorum)* als erfüllt, wenn eine Teilmenge aller Knoten den Schreibzugriff quittieren. Quoren können somit ein Maß für Datenintegrität darstellen. Häufig ist eine weitere Bedingung in einem auf Quoren basierendem System, dass sich Schreib- und Lesequorum sich *überschneiden (Overlap)*. Formal gilt in diesem System (siehe Abbildung 4.3[17]) bestehend aus N Knoten (Replikate), dem Lesequorum R und dem Schreibquorum W:

- Es liegt ein Overlap vor, wenn $R + W > N$.
- Optimale Lesezugriffe und langsame Schreibzugriffe erreicht man mit dem Read-One-Write-All-Quorum (ROWA), weil für eine Leseoperation nur ein Knoten der Schnittmenge angehören muss, jedoch für eine erfolgreiche Schreiboperation alle Knoten dies quittieren müssen (links in Abbildung 4.3).
- Ein Mehrheitsquorum für einen Schreibzugriff ist erreicht, wenn gilt $W \geq \frac{N}{2} + 1$ (rechts in Abbildung 4.3) und im Falle eines Lesezugriffs gilt $R \geq \frac{N}{2} + 1$ (mittig in Abbildung 4.3). Für die Kombination beider Fälle gilt $W \geq \frac{N}{2} + 1 \leq R$.
- Es liegt kein Overlap vor, wenn $R+W \leq N$ gilt. So spricht man von einem partiellen Quorum. In diesem Fall gilt eine starke Datenkonsistenz als nicht erfüllt.[18]

Eine Möglichkeit zur Ermittlung der Latenz während der Replikation und folglich deren Auswirkung auf die Datenkonsistenz stellen Patil et al. [Pat+11] vor: In einem System bestehend aus mehreren Knoten wird ein Datensatz mit bekannten Werten zentral aktualisiert. Da die einzelnen Knoten ebenfalls bekannt sind, werden die Knoten direkt asynchron angesprochen und verglichen. Dadurch wird ermittelt, ob der zurückgelieferte Wert mit dem bekannten aus der Aktualisierung übereinstimmt. Dieser Test wird in zeitlichen Intervallen auf allen Knoten ausgeführt und daher als paralleler Test (Parallel Testing) bezeichnet. Es wird also überprüft, wie lange veraltete Daten (Stale Read) zurückgegeben werden, um daraus auf die Leistungsqualität zu schließen.[19]

[16] Vgl. [Wie15, S. 295ff]

[17] Abbildung entnommen aus `http://www.inf.ufpr.br/aldri/disc/slides/SD712_lect12.pdf` (besucht am 31.03.2017)

[18] Vgl. [Wie15, S. 298f]

[19] Vgl. [Pat+11]

4.1.4 Transaktionalität

Die Leistungsmerkmale einer NoSQL-Datenbank bezüglich der Unterstützung von Transaktionen lassen sich untersuchen, indem man den *Durchsatz (Throughput)*, dies ist die Anzahl ausgeführter Datenoperationen pro Zeiteinheit, erhebt und ein Maß für die Datenkonsistenz bildet. Für letzteres setzt man die bei der Ausführung der Arbeitslast aufgetretenen Anomalien in Bezug zur Anzahl ausgeführter Datenoperationen setzt.[20] Hierfür stellen Dey et al. [Dey+14] ein anwendungsspezifisches Testverfahren für Applikationen vor, deren Datenzugriff dem Muster WORM („Write-Once-Read-Many“) folgen.

Die Tests der Transaktionsunterstützung nach Dey et al. im YCSB+T-Framework orientieren sich an den TPC-Benchmarks und simulieren den idealisierten Geschäftsfall eines geschlossenen Wirtschaftskreislaufs. Der Ablauf eines Tests gliedert sich in drei Phasen, wovon die ersten beiden analog im TPC-Benchmark zu finden sind und letztere spezifisch für nicht-relationale Datenbanksysteme ist:[21]

1. **Ladephase:** Als erstes wird die synthetische Datenmenge der Arbeitslast für den Test erzeugt und in die Datenbank eingelesen.

2. **Transaktionsphase:** Im Test erfolgt anschließend die Transaktionsphase, bei der die in der Arbeitslast generierten Transaktionsabfolgen ausgeführt werden. Dabei besteht eine Transaktion jeweils immer aus einer Zusammensetzung von unterschiedlichen Datenoperationen, wie sie in Abschnitt 4.1.1 definiert und im YCSB-Framework standardisiert sind (siehe Abschnitt 3.2.1).

3. **Validierungsphase:** Nachdem die Transaktionsphase beendet wurde, erfolgt die Validierungsphase. Im implementierten Testmodell des geschlossenen Wirtschaftskreislaufs wird für jeden der einzelnen Teilnehmer eine bestimmte Summe vor und nach dem Test überprüft. Die dabei ermittelten Abweichungen werden in Bezug zur Anzahl ausgeführter Datenoperationen gesetzt. Das Ergebnis dieser Validierungsphase ist ein Vergleichswert; ein Wert von Null entspricht dabei einer vollkommenen Wahrung der Datenintegrität.[22]

4.1.5 Polyglotte und Multi-Modell-Systeme

Die Evaluierung von Einsatz und Leistungsfähigkeit polyglotter Systeme wird getrieben von der Annahme, dass komplexe Datenstrukturen sich am ehesten durch ein adäquates Datenmodell abbilden lassen und somit eine bestmögliche Verarbeitungsgeschwindigkeit bezüglich Datenmenge und Datenstrukturen erst durch den Einsatz eines dedizierten Datenbanktyp erreicht werden kann. Dieser Ansatz ist evident, weil aufgrund der bisherigen technologischen Entwicklung relationale Datenbanksysteme vorherrschend waren und somit oft Datenstrukturen in ein relationales Datenmodell überführt wurden. Dies führt zu einer höheren Komplexität im Datenmodell und ist mit Nachteilen bei der Verarbeitungsgeschwindigkeit sowie der Implementierung und Wartung solcher Systeme behaftet.[23]

Ein Experiment von Fioravanti et al. [Fio+16] untersuchte die Leistungseigenschaften eines Systems, dessen Persistenzschicht bestehend aus einem RDBMS und einem Objekt-relationalem Mapping-Framework (ORM) migriert wird zu zwei unterschiedlichen nicht-relationalen Datenmodellen und Datenbanktypen (Dokument- und Graphdatenbank). Die

[20] Vgl. [DFR15]
[21] Vgl. [Dey+14]
[22] Vgl. [Dey+14]
[23] Vgl. [Fio+16]

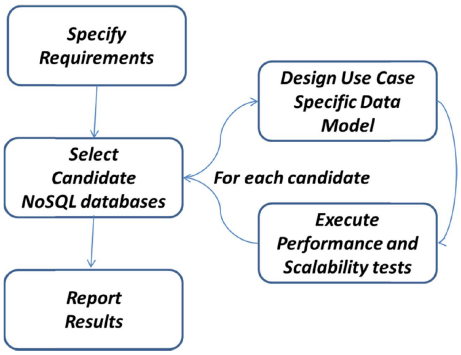


Abbildung 4.4: Vorgehensmodell „Lightweight Evaluation and Prototyping for Big Data“

Studie erfolgt am Beispiel eines Bestandssystems und unterscheidet sich aufgrund dessen im Vorgehen in bemerkenswerter Weise von den bisher vorgestellten Konzepten.

- **Vorgehensmodell:** Das praktische Vorgehen orientiert sich an dem iterative Vorgehensmodell *„Lightweight Evaluation and Prototyping for Big Data (LEAP4BD)“* von Klein et al. [Kle+15]. Dieses Modell (dargestellt in Abbildung 4.4[24]) kann als eine Vereinfachung des in Abschnitt 3.1.4 vorgestellten Verfahrens der „Cloud Evaluation Experiment Methodology (CEEM)“ gesehen werden.

- **Datenmodell:** Eine bereits vorliegende Beschreibung der Anwendungsfälle und verwendeten objektorientierten Domänen auf Basis eines Meta-Modells ermöglicht diesem Ansatz eine einfache Überführung in das und Anwendung im Datenmodell der zu testenden Dokument- und Graphdatenbank.

- **Datenanalyse:** Der Studie liegt die Annahme zugrunde, dass die Schreib- und Lesegeschwindigkeit der Datensätze prinzipiell unabhängig vom Datenbanktyp ist, sondern von der Komplexität der Datensatzzusammensetzung abhängt. Deswegen wurden die Bestandsdaten einer Datenanalyse unterzogen, um Aufschluss über ihre Zusammensetzung zu erhalten.

- **Arbeitslast:** Auf Basis der gewonnenen Erkenntnisse über die Zusammensetzung der existierenden Datenmenge wurde je eine realitätsnahe und eine künstliche Arbeitslast erzeugt. Die synthetisch erzeugte Arbeitslast dient der Evaluation des Leistungsverhaltens einer Lösung in Extremform (Stresstest).

Die Leistungsanalyse zu Multi-Modell-Datenbanksystemen [OVC16], welche mehrere Datenmodelle in einer Implementierung über eine gemeinsame Schnittstelle nutzbar machen, vergleicht zwei solcher Implementierungen mit zu Datenbanktypen, die ausschließlich ein Datenmodell unterstützen. In diesem Fall sind es zwei Multi-Modell-Systeme, die das Dokument- und Graphdatenmodell unterstützen, und jeweils ein System, welches nur eins dieser beiden Datenmodelle anbietet. Bei der Arbeitslastgenerierung wurde ein Algorithmus verwendet, der für die Graphdatenbanken ein skalenfreis Netzwerk[25] erzeugt hat, um so eine Vergleichbarkeit hinsichtlich der Beschaffenheit des Graph zu gewährleisten. Der Testablauf in dieser Studie gliedert sich in eine Lade- und eine Ausführungsphase.[26]

[24] Abbildung entnommen aus [Kle+15]

[25] In einem *skalenfreien Netzwerk* ist die Anzahl von Verbindungen pro Knoten nach einem Potenzgesetz verteilt. Ein Potenzgesetz beschreibt die Abhängigkeit zwischen zwei Größen – in diesem Fall Knoten und Kanten des Graphen.

[26] Vgl. [OVC16]

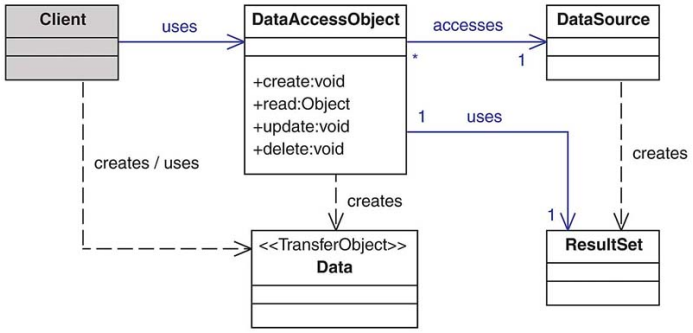


Abbildung 4.5: Vorlage für eine Schnittstelle (API)

4.2 Durchführung

Die Durchführung einer Leistungsanalyse, bei der unterschiedliche Datenmodelle und Datenbanktypen unter Verwendung einer synthetischen und standardisierten Arbeitslast verglichen werden, birgt die Gefahr, dass Elemente verglichen werden, die scheinbar nicht vergleichbar sind. Dies betrifft neben den in Abschnitt 4.1.5 genannten Beispielen auch ein Benchmark zwischen einer relationalen und nicht-relationalen Datenbank. Eine Vergleichbarkeit von verschiedenen, skalierten Testumgebungen kann aber dadurch gegeben sein, dass das Vorgehen und die Testumgebung standardisiert sind.[27] Ein standardisiertes Vorgehen ist insbesondere beim Datenzugriff und den Datenoperationen notwendig, weil diese je nach Datenbanktyp und Implementierung unterschiedlich sind. Während in SQL-Datenbanken Datenoperationen dem CRUD-Schema folgen, liegt bei NoSQL-Datenbanken keine einheitliche Konvention vor (siehe Abschnitt 4.1.1).

Deswegen bedient man sich einer Schnittstelle, mit der die spezifischen Implementierungen gekapselt sind.[28] Die Architektur einer solchen Schnittstelle (API) für ein Datenzugriffsobjekt (DAO) zeigt das UML-Klassendiagramm in Abbildung 4.5. Ferner ist eine weitere Funktion dieser Schnittstelle die Kapselung des Datenzugriffs, da dieser ebenfalls mit dem Datenbanktyp variiert. Beispiele hierfür sind API-Zugriffe via SQL, REST, Javascript oder in einer nativen Programmiersprache.[29] In bestimmten Fällen, wie dem Vergleich von Datenbanken in Bezug auf den Umgang mit GIS-Daten, kann man die Nutzung einer generalisierten Schnittstelle zugunsten einer direkten Nutzung der datenbankspezifischen API aufgeben, wenn diese jeweils äquivalente Operationen und Methoden anbieten oder gerade die Leistungsfähigkeit der Schnittstelle analysiert werden soll.[30]

4.2.1 Äquivalenz von Informationen

Ein Leistungsvergleich über verschiedene Datenmodelle und Datenbanktypen hinweg erfordert, dass sie zueinander in bestimmten Aspekten äquivalent sind. Die verschiedenartigen Datenmodelle beeinflussen die Datenstruktur. Sie haben jedoch dieselbe Informationskapazität und können dadurch mithilfe von Äquivalenz-Definitionen und Hierarchien von Äquivalenzen in ein übergeordnetes Modell überführt werden. Zwei Datenstrukturen gelten als äquivalent bezüglich ihrer Informationskapazität, wenn der Zustandsraum gleich ist. Dies liegt genau dann vor, wenn für alle Eigenschaften und Werte der einen

[27] Vgl. [Kle+15]
[28] Vgl. [Coo+10], [LM13] u.a.
[29] Vgl. [ABF14b], [Pat+11]
[30] Vgl. [SGR15]

Datenstruktur eine Zuordnung (Mapping) in die andere Datenstruktur existiert. Dies gilt auch für die Beschreibung der Datenstruktur selbst.[31]

Für einen Leistungsvergleich reicht bereits eine Äquivalenz der Datenmodelle bezüglich der Sicht (Präsentation) aus, wenn mittels Abfrage aus beiden Datenmodellen dieselben Informationen geliefert werden. Dieser Maßstab hebt hervor, dass beide Datenmodelle die gleichen Informationen abbilden unabhängig ihrer Gemeinsamkeiten und Unterschiede. Somit gelten zwei Sichten eines Domänenmodells unabhängig von der Datenstruktur und den verwendeten Datentypen als äquivalent, wenn die darin enthaltenen Informationen auf Datenebene nach Serialisierung in eine Zeichenkette identisch sind.[32]

4.2.2 Beschreibung des experimentellen Testaufbaus

Jede experimentelle Untersuchung beinhaltet eine kurze Beschreibung des Testaufbaus. Für eine Leistungsmessung nach dem in Abschnitt 3.1.4 vorgestellten CEEM-Verfahren empfiehlt sich ein Verzeichnis zur Identifizierung aller Abhängigkeiten und Zusammenhänge zwischen den Parametern und Variablen. Diese Zusammenhänge werden auch Faktoren genannt werden.[33] Ein solches Verzeichnis wird beispielhaft in Tabelle 4.2 gezeigt. Entsprechend dem Vorgehen der Cloud Service Evaluation sind in der Testumgebung und dem SUT alle physikalischen Eigenschaften in Form von Parameter und Variablen eines zu testenden Datenbanksystems aufgeführt. Die Arbeitslast kann nach Definition des Benchmarking-Frameworks (siehe Abschnitt 3.1.2) zur Testumgebung gezählt werden, ist jedoch außerhalb des SUT angesiedelt.[34] In der Tabelle 4.3 sind alle Parameter und Variablen betreffend der Arbeitslast aufgelistet. Die in den Tabellen aufgeführten Daten stammen aus der in dieser Arbeit verwendeten Literatur.[35]

[31] Vgl. [Fio+16]
[32] Vgl. [Fio+16]
[33] Vgl. [LOZ13]
[34] Vgl. [Bou+10]
[35] Aus Gründen der Übersichtlichkeit ist auf die Quellenangabe zu den einzelnen Parameter verzichtet worden, da sie bereits in den vorhergehenden Abschnitten erläutert wurden.

Bezeichnung	Beschreibung	Beispielwert
Testumgebung		
Knoten	Anzahl genutzter Knoten	9
Verteilung Knoten	Angaben zur Netzwerktopologie	3 Rechenzentren, geografisch verteilt
Virtuelle Maschine	Betrieb des Systems in einer virtualisierten Umgebung	ja, nein
Prozessor (CPU)	Anzahl der Prozessoren/ Prozessorkerne	8
Prozessorgeschwindigkeit	Prozessorgeschwindigkeit in Gigahertz	3,2 GHz
Arbeitsspeicher	Größe des verwendeten Arbeitsspeichers	10 Gigabyte
Betriebssystem	Bezeichnung Betriebssystem, Version, Bit-Version	Ubuntu 17.10 (64 Bit)
Netzwerk	Netzwerkeigenschaften	Gigabit Ethernet
Speichersystem	Angaben zu Speichersystem	HDD, SSD, RAM
System unter Test (SUT)		
Datenbanktyp	zugrundeliegendes Datenmodell	Schlüssel-Wert-Datenbank
Datenbankimplementierung	Name und Version	Cassandra 3.10
Partitionierung	Anzahl der Partitionen (Shards)	3
Replikationen	Anzahl der Replikationen	3
Replikationsmodus	Angabe zur Art der Datensynchronisation	asynchron
Quorum Lesen	Konfiguration des Lesequorums bezüglich Anzahl der Knoten	R_N
Quorum Schreiben	Konfiguration des Schreibquorums bezüglich Anzahl der Knoten	W_N
Threads	Anzahl gleichzeitiger Benutzer (Clients) oder Prozesse (Threads)	128

Tabelle 4.2: Verzeichnis der physikalischen Eigenschaften – Testumgebung und SUT

Bezeichnung	Beschreibung	Beispielwert
Arbeitslast		
Datenoperationen	Auswahl der Datenoperationen	Read, Insert, Update, Scan
Anzahl der Datenoperationen	Anzahl der auszuführenden Datenoperationen pro Arbeitslast	10000
Zusammensetzung Datenoperationen	Angabe der prozentualen Zusammensetzung bezüglich der verwendeten Datenoperationen	50% Read, 50% Write
Anzahl der Datenoperationen	Anzahl der auszuführenden Datenoperationen pro Arbeitslast	10000
Anzahl der Datensätze	Anzahl der Datensätze in der Datenbasis respektive Arbeitslast	600000
Datenstruktur	Anzahl der Felder in einem Datensatz, Beschreibung der Datenstruktur eines Datensatzes	10
Größe des Datenfeldes	Größe des Datenfeldes in Bytes oder Kilobyte	100 Bytes, 1 Kilobyte
Größe des Datensatzes	Größe des Datensatzes als Summe aller einzelnen Datenfelder	10 Kilobyte
Datenauswahl	Wahrscheinlichkeit für Datensätze bezüglich Auswahl für Datenoperation	Zipfsche Verteilung
Transaktionen	Anzahl auszuführender Transaktionen	10000

Tabelle 4.3: Verzeichnis der Parameter und Variablen – Arbeitslast

5 Leistungsbewertung

Die Leistungsanalyse und anschließende -bewertung im Sinne eines Leistungsvergleichs von NoSQL-Datenbanksystemen orientiert sich an folgenden vier Kriterien:[1]

- Die **Relevanz** für die zu untersuchende Domäne ausgeprägt in Form einer realitätsbezogenen Testumgebung und System unter Test sowie einer anwendungsnahen Arbeitslast.
- Die **Portabilität** auf die verschiedenen Datenbankimplementierungen hinsichtlich des Datenmodells und der Arbeitslast.
- Die Fähigkeit zur **Skalierung** der eingesetzten Werkzeuge für den Leistungsvergleichs an massiv parallelen Systemen.
- Ein **eindeutiges Verständnis** der erhobenen Rohdaten und berechneten Metriken.

Ferner ist zu beachten, dass die durchgeführten Leistungsbewertungen und Leistungsvergleiche hinsichtlich der Methodik und den eingesetzten Werkzeugen kritisch zu hinterfragen sind.

5.1 Rohdaten und Metriken

Aus dem Modell der Cloud Service Evaluation ist bekannt, dass eine experimentell untersuchte Leistungseigenschaft die Zuordnung von einer physikalischen zu einer kapazitiven Einheit ist.[2] In der Phase der Leistungsmessung im experimentellen Versuch werden an einer physikalischen Eigenschaft sogenannte Rohdaten zu einer primären Kapazitätseigenschaft erhoben.[3] Diese Rohdaten sind die Grundlage für die Leistungsbewertung und die Berechnung der sekundären Kapazitätseigenschaften. Sie werden innerhalb eines Leistungsvergleichs auch als Metriken bezeichnet. Die Tabelle 5.1 zeigt eine Auflistung möglicher Rohdaten und Metriken.[4]

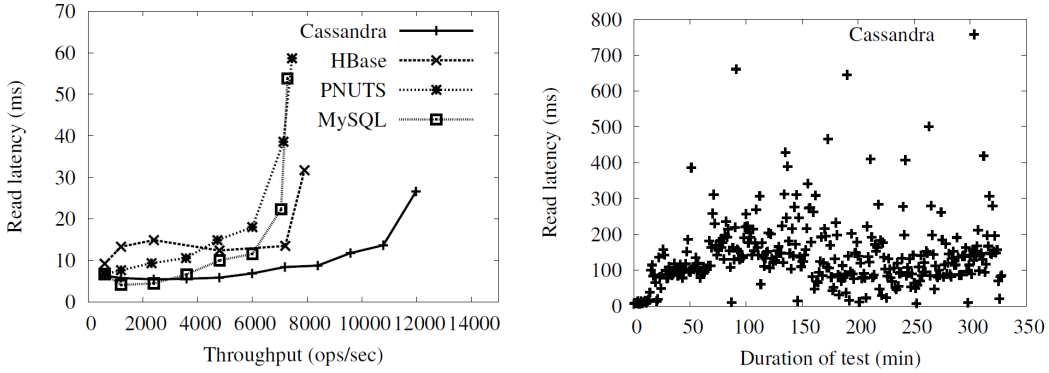
5.1.1 Quantitative Analyse

Eine quantitative Leistungsanalyse und der darauf beruhende Leistungsvergleich erfolgt auf Basis einer definierten Arbeitslast und Datenmenge für die zu testenden NoSQL-Datenbanksysteme.[5] Als erstes bietet es sich an, die gemessene Verarbeitungszeit einer Arbeitslastlösung zu vergleichen. So erhält man einen ersten Überblick über das Leistungsverhalten.[6] Hieraus ergibt sich auch die Verarbeitungsgeschwindigkeit, sprich die Anzahl

[1] Vgl. [Coo+10]
[2] Vgl. [LOR16]
[3] Siehe hierzu auch Abschnitt 3.1.3 und Tabelle 3.2
[4] Aus Gründen der Übersichtlichkeit ist auf die Quellenangabe zu den aufgeführten Rohdaten und Metriken verzichtet worden. Sie stammen aus der in dieser Arbeit verwendeten Literatur und sind in anderen Abschnitten erläutert.
[5] Vgl. [SE16]
[6] Vgl. [ABF14b]

Bezeichnung	Beschreibung	Beispielwert
Rohdaten		
(Durchschnittliche) Ausführungszeit	Zeiteinheit in Millisekunden; Angabe pro Arbeitslast oder Anzahl ausgeführter Operationen	1038 ms
Durchläufe	Anzahl der Durchläufe pro Arbeitslast oder ausgeführter Operationen	5
Clients	Anzahl genutzter Clients	100
Threads	Anzahl verwendeter Threads	20
Zeitversatz der Abfragen	Zeiteinheit in Millisekunden; Angabe zu Zeitversatz zwischen den Abfragen	simultan, 1000
(Durchschnittliche) Auslastung CPU	Werte für ein Zeitintervall	0–100%
Arbeitsspeicherverbrauch	Prozentuale Werte für Zeitpunkte in einem Zeitintervall	0–100%
Festplattenspeicherverbrauch	Prozentuale Werte für Zeitpunkte in einem Zeitintervall	0–100%
Netzwerkaktivität	Bytes pro Sekunde	1024
Metriken		
(Durchschnittliche) Ausführungsgeschwindigkeit	Zeiteinheit in Millisekunden; Angabe pro Durchläufe und Arbeitslast oder Anzahl ausgeführter Operationen/Transaktionen	4711 ms
Latenz	Zeiteinheit in Millisekunden	250 ms
Erfolgreiche Transaktionen	Anzahl erfolgreich ausgeführter Transaktionen	9950
Fehlgeschlagene Transaktionen	Anzahl ausgeführter Transaktionen, die fehlgeschlagen sind	50
Abweichungen Transaktionen	Verhältnis fehlgeschlagener Transaktionen zu allen ausgeführten Transaktionen in Arbeitslast	0,05
Speed Up	Kennzahl, die das Verhältnis von Ausführungszeit/ Verarbeitungsgeschwindigkeit zur Anzahl der Knoten beschreibt	350
Scale Up	Kennzahl, die das Verhältnis von Daten und Last zur Anzahl Knoten beschreibt	0,5

Tabelle 5.1: Verzeichnis der Rohdaten und Metriken



(a) Liniendiagramm (b) Punktdiagramm

Abbildung 5.1: Diagrammtypen zur Darstellung von Metriken

ausgeführter Datenoperationen oder verarbeiteter Datensätze pro Zeiteinheit (meistens Sekunden). Ein detaillierteres Bild ergibt sich, wenn mehrere Arbeitslasten mit jeweils unterschiedlichen Datenoperationen definiert und die Anzahl ausgeführter Datenoperationen je Arbeitslast ansteigend variiert werden. Mit diesem Vorgehen lassen sich Unterschiede hinsichtlich der Verarbeitungsgeschwindigkeit einer Datenbankimplementierung bei steigender Last ermitteln.[7]

Das mehrmalige (meistens fünfmalige) Ausführen einer Arbeitslast dient einer besseren Qualität der Rohdaten. Daraufhin wird für jede zu testende Lösung ein Durchschnittswert gebildet und für die weiteren Berechnungen von Metriken herangezogen.[8] Bei der Berechnung von Durchschnittswerten ist darauf zu achten, ob ein arithmetische Mittel oder ein Median herangezogen wird. Das arithmetische Mittel $\bar{x}$ ist für n Messwerte definiert als:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Im Gegensatz zum arithmetischen Mittel ist der Median robuster gegenüber Ausreißern bei Zahlenwerten. Der Median $\tilde{x}$ ist für n sortierte Messwerte definiert als:

$$\tilde{x} = \begin{cases} x_{\frac{n+1}{2}} & n \text{ ungerade} \\ \frac{1}{2}\left(x_{\frac{n}{2}} + x_{\frac{n}{2}+1}\right) & n \text{ gerade} \end{cases}$$

Eine sehr häufig genutzte Kennzahl zur Leistungsbewertung ist – neben der Messung der absoluten Ausführungszeit und der Ermittlung von Durchschnittswerten – die Ermittlung des Datendurchsatzes in ausgeführten Datenoperationen oder Transaktionen pro Sekunde. Diese Kennzahl wird dann verwendet, wenn das Leistungsverhalten in unterschiedlichen SUT-Konfigurationen gestestet und analysiert werden soll. Beispielsweise können hierfür Variation der Anzahl, der von der Datenbank genutzten Knoten, Clients, Threads oder Benutzer sowie den Zeitversatz zwischen einzelnen Datenabfragen (Requests).[9] Die Aufarbeitung der gemessenen Daten und berechneten Metriken erfolgte in der vorliegenden Literatur zuerst in tabellarischer und dann in grafischer Form (siehe Abbildung 5.1[10] anhand von Linien-, Säulen-, Balken- und Punktdiagrammen).

[7] Vgl. [LM13], [OVC16], [SE16]
[8] Vgl. [SGR15], [LM13]
[9] Vgl. [Dey+14], [Kle+15], [SE16], [SGR15]
[10] Abbildungen entnommen aus [Coo+10]

5.1.2 Skalierbarkeit

Die Fähigkeit zur Skalierbarkeit eines Systems lässt sich an vier Aspekten betrachten, wie sie in Abschnitt 2.2.4 vorgestellt wurden. Für die Leistungsbewertung von NoSQL-Datenbanksystemen ist die Skalierbarkeit bezüglich der Last und der Daten relevant. Zu dieser Betrachtung gehört auch die Analyse und Bewertung des Leistungsverhaltens bei horizontaler (Scale Out) und vertikaler (Scale Up) Skalierung oder der Kombination beider Ansätze in Form von Elastizität. Die strukturelle Skalierbarkeit wird als gegeben betrachtet aufgrund der Möglichkeiten, weitere Ressourcen in Form von Knoten hinzuzufügen und den Mechanismen zur Datenpartitionierung und -replikation.

Die Datenskalierbarkeit, die auch zeitlich-räumliche Skalierbarkeit genannt wird, lässt sich untersuchen, indem man die Ausführungszeit pro ausgeführter Arbeitslast misst und anhand dessen die Verarbeitungsgeschwindigkeit errechnet.[11] Das Leistungsverhalten bei steigender Datenmenge lässt sich Durch Variation der gesamten Datenmenge untersuchen. Die Gesamtdatenmenge, auf denen die Datenoperationen ausgeführt werden, wird jeweils um eine 10er-Potenz (1 GB, 10 GB, 100 GB) erhöht. Eine weitere Möglichkeit, das Verhalten bei Datenskalierung zu evaluieren, ist möglich, indem man die Anzahl der betroffenen Datensätze variiert, die pro definierter Datenoperation erstellt, gelesen, geändert oder gelöscht werden.[12] Dies kann durch Konfiguration der Arbeitslast im YCSB-Framework erreicht werden (siehe Abschnitt 4.1.2).

Ein weiterer Zusammenhang zwischen der Datenskalierbarkeit und dem Hinzufügen weiterer Ressourcen – bei NoSQL-Datenbanken sind das typischerweise Knoten – ist der Grad der Beschleunigung oder Parallelisierung (Speed Up). Hierbei wird die Ausführungszeit und Verarbeitungsgeschwindigkeit für die gleiche Arbeitslast in Bezug zur Anzahl der Knoten im SUT verglichen. Zusätzlich kann ein möglicher Mehraufwand (beispielsweise möglicher Overhead durch Netzwerkkommunikation und Synchronisation der Knoten) in der parallelen Verarbeitung dadurch aufgezeigt werden. Der Grad der Parallelisierung korrespondiert nicht zwangsläufig mit einer kürzeren Ausführungszeit oder zumindest nicht in einer höheren Verarbeitungsgeschwindigkeit um den Faktor hinzugenommener Knoten.[13]

Eine Lastskalierbarkeit liegt bei einem Datenbanksystem vor, wenn es ohne spürbare Verzögerung mehr Prozesse (Threads) bedienen kann. Diese Eigenschaft lässt sich dadurch erforschen, dass die Arbeitslast jeweils von jedem einzelnen Prozess, deren Anzahl zu definieren ist, ausgeführt wird. Dies geschieht im YCSB-Framework über Parameter beim Generieren der Arbeitslast. Ferner lässt sich bestimmen, mit welchem Zeitversatz die Prozesse während des Tests gestartet werden sollen. Hierbei wird die Ausführungszeit oder Verarbeitungsgeschwindigkeit beobachtet. In vorliegenden Studien lassen sich für die verschiedenen Datenbankimplementierungen sehr unterschiedliche Ergebnisse diesbezüglich finden.[14]

Mit der Kombination aus den Tests zu Daten- und Lastskalierbarkeit lässt sich die Gesamtskalierung, das Scale Up, untersuchen. Bei diesem Test wird eine Arbeitslast mit jeweils einer Kombination (bestehend aus einer Anzahl Knoten und einer definierten Datenmenge für eine unterschiedliche Anzahl Threads) ausgeführt, die Ausführungszeit gemessen und die Verarbeitungsgeschwindigkeit erhoben.[15] Erstrebenswert ist ein gleichbleibender, linearer Verlauf bei steigender Anzahl gleichzeitig ausgeführter Threads, wie die Abbildung 2.5(b) auf Seite 12 zeigt.

[11] Vgl. [ABF14b], [ABF14a], [YD16]
[12] Vgl. [ABF14a]
[13] Vgl. [ABF14a], [SE16], [OVC16]
[14] Vgl. [ABF14a], [Kle+15], [Pat+11]
[15] Vgl. [ABF14a]

Die Leistungsbewertung der Elastizitätsfähigkeit, dem störungsfreien Hinzufügen weiterer Knoten zum System und Ressourcen in Form von CPU-Leistung oder Speicher zum Knoten erfolgt in einer zeitlichen Dimension. Dabei wird die Arbeitslast für eine fixe Datenmenge wiederholend über einen Zeitraum von beispielsweise 60 Minuten ausgeführt. Der Test startet mit einem Knoten und alle fünf Minuten wird ein weiterer Knoten oder weitere Ressourcen zum Knoten hinzugeschaltet sowie die Ausführungszeit und Verarbeitungsgeschwindigkeit ermittelt. Hier ist insbesondere das Verhalten zum Zeitpunkt der Bereitstellung des nächsten Knotens oder der weiteren Ressource interessant, weil die Höhe der Latenz, sprich der Ausführungszeit in diesem Moment, die Bemessungsgrundlage für einen störungsfreien Betrieb ist.[16]

5.1.3 Datenkonsistenz

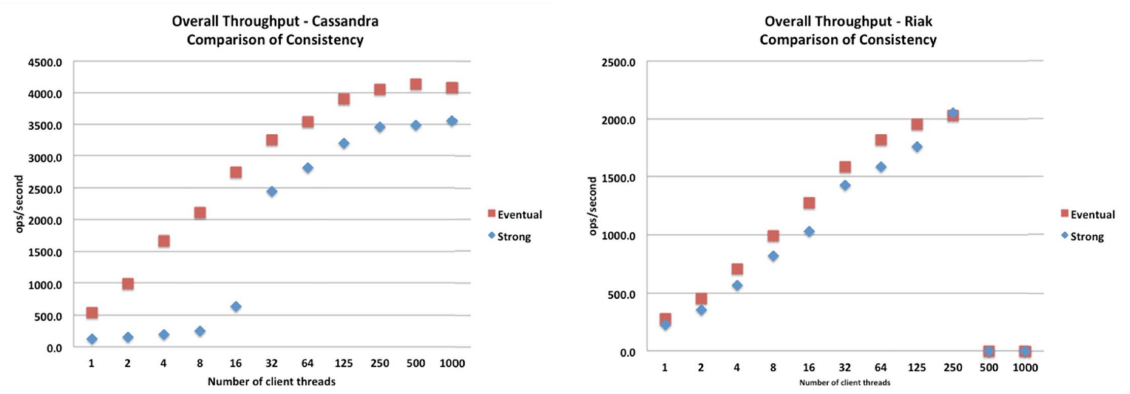
Der geforderte Grad an Datenkonsistenz spielt eine entscheidende Rolle bei der Dimensionierung der Hardware eines Datenbanksystems. Im Mittelpunkt dieser Betrachtung steht der Vergleich zwischen einem System bestehend aus einem vertikal skalierten Einzelknoten einerseits und einem weiteren bestehend aus mehreren horizontal skalierten Knoten andererseits. Die Parametrisierung der Datenkonsistenz durch Quoren für die lesenden und schreibenden Datenoperationen erlaubt Rückschlüsse auf das Leistungsverhalten und den zu erwartenden kommunikativen Mehraufwand, welcher durch die Synchronisation der Partitionen und Replikationen (Knoten) entsteht.

In der Fallstudie von Klein et al. [Kle+15] wurde das Laufzeitverhalten unterschiedlicher NoSQL-Datenbanktypen unter Berücksichtigung der Quorendefinition hinsichtlich der Auswirkungen einer starken und einer schwachen respektive eventuellen Datenkonsistenz (Eventual/ Weak Consistency) betrachtet. Für jeweils ein Typ von Datenbanksystem, welche aus einer Serverkonfiguration bestehend aus einem vertikal skalierten Einzelknoten. Dies wurde mit einem horizontal skalierten Knoten verglichen. Die Quoren dienten hierbei als Definitionsmerkmal für eine starke oder mögliche Datenkonsistenz. Ferner wurde zwischen verschiedenen Arbeitslasten unterschieden: nur lesend oder nur schreibend sowie schreibend und lesend. Weiterhin ist für die Leistungsbewertung entscheidend, wie das Verhalten bei Last ist, wenn die Anzahl ausführender Threads durch die Arbeitslast skaliert wird. In diesem Zusammenhang wird mit dem Begriff Latenz die Ausführungszeit oder Verarbeitungsgeschwindigkeit bezeichnet, mit der die Datenoperationen ausgeführt werden. Die Berechnung der Latenz erfolgt dabei in Abhängigkeit zur Anzahl der simultanen Threads. Die Autoren der Fallstudie verwenden bei der Bildung der Durchschnittswerte für die Ausführungszeit das arithmetische Mittel und ein 95-%-Quantil, um extreme Ausreißer herauszufiltern.[17] Der Mittelwert x_p ist einem p-Quantil ist für n Messwerte definiert als:

$$x_p = \begin{cases} \frac{1}{2}(x_{(np)} + x_{(np+1)}) & np \text{ ganzzahlig} \\ x_{(\lfloor np \rfloor + 1)} & np \text{ nicht ganzzahlig} \end{cases}$$

In der Studie wurden zwei Schlüssel-Wert-Datenbanken (Cassandra[18] und Riak[19]) hinsichtlich ihrer Verarbeitungsgeschwindigkeit unter Verwendung starker oder eventueller Datenkonsistenz bei ansteigender Last untersucht. Die Abbildungen 5.2(a) und 5.2(b)[20] zeigen jeweils für einen Datenbanktyp den Durchschnittswert für den Datendurchsatz

[16] Vgl. [Coo+10]
[17] Vgl. [Kle+15]
[18] `http://cassandra.apache.org`
[19] `http://docs.basho.com`
[20] Abbildungen entnommen aus [Kle+15]



(a) Datenbank: Cassandra

(b) Datenbank: Riak

Abbildung 5.2: Vergleich der Verarbeitungsgeschwindigkeit bezüglich der Datenkonsistenz

aus den drei Arbeitslasten (nur schreibend, nur lesend, schreibend und lesend) in Bezug zur Anzahl simultaner Prozesse. Die Datenbankimplementierung Cassandra hat in diesem Leistungsvergleich, welcher mit YCSB-Framework durchgeführt wurde, einen um bis zu Faktor zwei höheren Datendurchsatz. Das Riak-System konnte im Gegensatz hierzu keine 500 oder gar 1000 Threads gleichzeitig bedienen.[21]

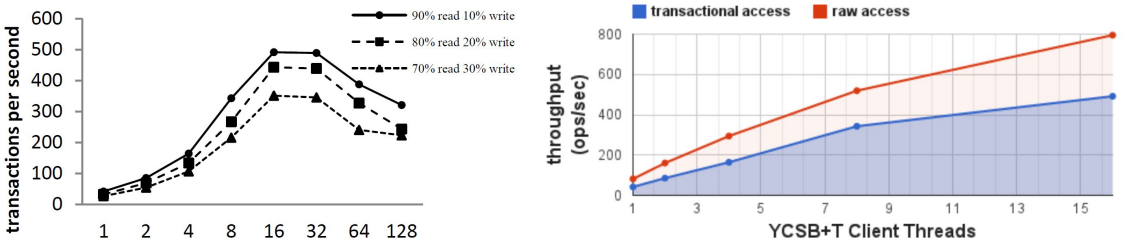
5.1.4 Validierung von Transaktionen

Die Umsetzung von Transaktionen ist mithin der anspruchsvollste Aspekt in NoSQL-Datenbanksystemen, weil oftmals keine oder nur eine sehr rudimentäre Unterstützung von Transaktionen durch die Datenbanken selbst angeboten wird. In diesem Kontext bezieht sich die Leistungsbewertung dann mehr auf die individuelle Lösung als auf den verwendeten Datenbanktyp. Es gilt hierbei anhand der in Abschnitt 2.2.3 vorgestellten zwei Implementierungsvarianten einer Transaktion zu unterscheiden.

Beim Ansatz der Transaktionsunterstützung über mehrere Datensätze in verteilten, heterogenen Schlüssel-Wert-Datenbanken wird der Test gegen die angebotene Schnittstelle ausgeführt, weil die fehlerfreie Transaktionsausführung als gegeben betrachtet wird: sie werden auf unterschiedlichen Systemen ausgeführt, die jeweils aus einem Einzelknoten bestehen. In diesem Fall erfolgt die Leistungsbewertung anhand der Betrachtung der Verarbeitungsgeschwindigkeit (Transaktionen pro Sekunde) unter steigender Last. Die Abbildung 5.3(a)[22] zeigt eine solche Leistungsanalyse am Beispiel von drei unterschiedlichen Arbeitslasten, die in ihrer prozentualen Zusammensetzung aus Lese- und Schreiboperation variieren.[23]

Das Testverfahren aus dem YCSB+T-Framework verfolgt einen anderen Ansatz, weil hier die fehlerfreie Transaktionsausführung als nicht gegeben betrachtet wird. Es dient vielmehr der Ermittlung des Grades an Fähigkeit zur transaktionalen Unterstützung. Auch in diesem Fall wird die Verarbeitungsgeschwindigkeit (Operationen pro Sekunde) unter steigender Last am Beispiel von drei unterschiedlichen Arbeitslasten, die in ihrer prozentualen Zusammensetzung aus Lese- und Schreiboperation variieren, betrachtet. Diese Transaktionen werden in Vergleich zu analogen, nicht als Transaktionen ausgeführten

[21] Vgl. [Kle+15]
[22] Abbildung entnommen aus [DFR15]
[23] Vgl. [DFR15]



(a) Verarbeitungsgeschwindigkeiten in [DFR15] (b) Verarbeitungsgeschwindigkeiten in [Dey+14]

Abbildung 5.3: Vergleich der Verarbeitungsgeschwindigkeiten unter Last bei Transaktionen

Datenoperationen gesetzt, wie in Abbildung 5.3(b)[24] dargestellt. In dem in Abschnitt 4.1.4 vorgestellten Testverfahren wird über die Arbeitslast ein geschlossener Wirtschaftskreislauf simuliert. Dieses Modell und das Vorgehen in Phasen ermöglicht eine Validierung der Transaktionen. Die Summe des Guthabens auf den Konten der Teilnehmern muss beim Test initial und final übereinstimmen, um ein Maß für die Wahrung der Datenintegrität zu bestimmen.[25] Diese Kennzahl γ, der *Simply Anomaly Score*, ist bei n Datenoperationen definiert als:

$$\gamma = \frac{|S_{initial} - S_{final}|}{n}$$

5.2 Vergleiche von unterschiedlichen Datenbanktypen

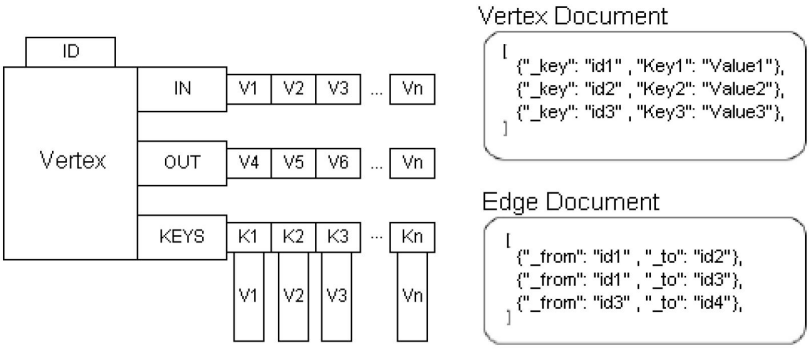
Der Leistungsvergleich von relationalen und nicht-relationalen Datenbanksystemen sowie von unterschiedlichen nicht-relationalen Implementierungen ist geprägt von den verschiedenartigen Datenmodellen und verwendeten Datenoperationen, die der Umsetzung des fachlichen Domänenmodells dienen.[26] Die Vergleichbarkeit ist unter solchen Bedingungen zu gewährleisten durch ein gleiches Vorgehen, eine standardisierte Arbeitslast, die Nutzung von Schnittstellen und der Äquivalenz von Informationen über ihre Präsentationssicht.[27] Die Leistungsbewertung in einem Vergleich von unterschiedlichen Datenbanktypen erfolgt anhand der in Abschnitt 5.1 aufgeführten Rohdaten und Metriken, einer quantitativen Analyse und der Betrachtung der Skalierungsfähigkeit, wie in den Abschnitten 5.1.1 und 5.1.2 dargelegt.

5.2.1 Graphdatenbanken

Im Zusammenhang mit dem Vergleich von unterschiedlichen Datenbanktypen sind insbesondere Graphdatenbanken interessant, weil diese unterschiedliche Konzepte einsetzen, um Knoten und Kanten sowie deren Eigenschaften abzubilden. Es werden zwei Konzepte zur Abbildung solcher Strukturen unterschieden, wie sie in Abbildung 5.4[28] dargestellt sind.

- **Indexfreie adjazente Struktur:** Die Abbildung 5.4(a) zeigt eine Struktur, bei der jeder Knoten eine Referenz auf die mit ihm über Kanten verbundenen weiteren Knoten besitzt. In diesem Modell wird kein Index zum Traversieren der Datenstruktur

[24] Abbildung entnommen aus [Dey+14]
[25] Vgl. [Dey+14]
[26] Vgl. [LM13], [SGR15], [OVC16]
[27] Vgl. [Coo+10], [Fio+16]
[28] Abbildung entnommen aus [OVC16]



(a) Indexfreie adjazente Struktur (b) Listen von Knoten und Kanten

Abbildung 5.4: Implementierungskonzepte von Graphen in Datenbanken

benötigt. Dies wirkt sich vorteilhaft auf die Verarbeitungsgeschwindigkeit aus, weil jeder Knoten seine adjazenten Verbindungen, welche unmittelbar mit ihm verbundenen Knoten, bereits kennt. Dieses Modell ist jedoch mit einem höheren Speicherverbrauch verbunden.[29]

- **Listen von Knoten und Kanten:** In Abbildung 5.4(b) sind zwei Listen dargestellt. Die Liste *Vertex* beinhaltet alle Knoten des Graphs und deren Eigenschaften, die Liste *Edge* hat diese Funktion für alle Kanten zwischen den Knoten. Die Indizierung in beiden Tabellen erfolgt mittels einer dedizierten Hash-Funktion.[30]

Die Leistungsbewertung einer zeitlich-räumlichen Skalierbarkeit von Graphdatenbanken ist von der Besonderheit geprägt, dass eine Partitionierung der Daten nicht oder nur sehr schwierig möglich ist. Folglich ist die Leistungsanalyse in diesem Fall eng mit der Datenstruktur und der Datenmenge der Arbeitslast verbunden. Dies betrifft die Anzahl von Knoten im Graphen, die bei der Ausführung der Arbeitslast zu bewältigen sind und die Höhe des Graphen, welcher traversiert werden muss. Für eine quantitative Betrachtung der Lastskalierbarkeit wird nun diese parametrisierte Arbeitslast mit einer unterschiedlichen Anzahl von Threads abgefragt und hinsichtlich der Ausführungsdauer bewertet.[31]

5.2.2 Empirische Analyse

Eine Leistungsbewertung kann auch auf Basis empirischer Verfahren erfolgen. Die Auswahl der Verfahren richtet sich nach der Fragestellung. In der leistungsbezogenen empirischen Analyse von Yassien und Desouky [YD16] wird für unterschiedliche Datenmodelle (relational, Spaltenfamilien, Schlüssel-Wert) der Zusammenhang einer größeren Last auf die Laufzeit, den Datendurchsatz und die Latenz untersucht. Die Autoren der Studie unterscheiden dabei Last in zwei Formen, nämlich in Anzahl auszuführender Datenoperationen sowie in Anzahl simultaner Threads. Für dieses Experiment wurden in drei unterschiedlichen Datenbanktypen jeweils 10 Millionen Datensätze eingeladen und mit dem YCSB-Framework vier Arbeitslasten ausgeführt, die sich in der Zusammenstellung ihrer Datenoperationen unterscheiden. Von jeder Arbeitslast existieren zwei Varianten, jeweils eine in Form auszuführender Datenoperationen sowie eine in Form von Threads.[32]

[29] Vgl. [Wie15, S. 53ff]
[30] Vgl. [Wie15, S. 56f]
[31] Vgl. [OVC16]
[32] Die Anzahl Threads ist in Intervallen von 1 bis 10 definiert. Für die Anzahl auszuführender Datenoperationen sind folgende Intervalle definiert: 1.000, 10.000, 100.000, 1.000.000, 10.000.000 und 15.000.000

Das angewandte statistische Verfahren ermittelt für jede Kombination aus Arbeitslast und Datenmodell die *Korrelationskoeffizienten* zur Berechnung des linearen Zusammenhangs zwischen der Anzahl von Datenoperationen oder Threads einerseits und Laufzeit, Datendurchsatz oder Latenz andererseits. Der Korrelationskoeffizient r ist für n Messwerte je unabhängige Variable X und Y definiert als:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2} \cdot \sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Beispiel: Wenn X die unabhängige Variable für die Anzahl von Threads und Y die unabhängige Variable für den Datendurchsatz ist, dann bedeutet ein Korrelationskoeffizient mit dem Wert:

- $r \approx 0$, dass die Anzahl der Threads und der Datendurchsatz keinen Zusammenhang aufweisen;
- $r > 0$ hat eine positive Korrelation; in diesem Fall gehen größere Werte von X einher mit größeren Werten von Y; dies heißt im Beispiel, dass mehr Threads im Zusammenhang mit einem höheren Datendurchsatz stehen;
- $r < 0$ hat eine negative Korrelation in diesem Fall hängen höhere Werte von X mit niedrigeren Werten von Y zusammen (und umgekehrt), was im obigen Beispiel bedeuten würden, dass mehr Threads mit einem niedrigeren Datendurchsatz korrelieren (und umgekehrt).

Ferner wird in der Studie die *statistische Signifikanz* für die Korrelation ermittelt und hierfür ein Signifikanzniveau von $\alpha = 0,05$ angenommen. In diesem Kontext wird auch die *Überschreitungswahrscheinlichkeit (p-Wert)* berechnet.[33]

In der Fallstudie von Fioravanti et al. [Fio+16] wird die Antwortzeit unterschiedlicher Persistenzlösungen für eine bereits existierende Anwendung empirisch untersucht. Die getesteten Lösungen variieren bezüglich des Datenmodells und Datenbanktyps (relational, dokument- und graphbasiert). Auf Basis einer vorhergehenden Datenanalyse des Bestandssystems wurde eine realitätsnahe sowie eine synthetische Datenmenge erzeugt. Die realitätsnahe Datenmenge besteht aus einer fixen Anzahl von Datensätzen, die sich in ihrer Zusammensetzung und Struktur erheblich unterscheiden. In der Arbeitslast wird nun jeder dieser Datensätze gelesen. Dieser Versuch wird 100 Mal wiederholt. Aus den gemessenen Daten berechnen die Autoren den Erwartungswert für die Antwortzeit in Millisekunden. Dieser Mittelwert wird zur Leistungsbewertung der einzelnen Lösungsvarianten herangezogen. Wenn X die Zufallsvariable für die Antwortzeit ist, dann ist der Erwartungswert μ definiert als:

$$\mu_X = \mathrm{E}(X) = \sum_i x_i \cdot P(X = x_i)$$

Ferner wird in dieser Analyse der Variationskoeffizient herangezogen als das relative Maß für die Streuung der Antwortzeit. Die Varianz V für eine Zufallsvariable X ist definiert als:

$$\mathrm{V(X)} = \sigma_X^2 = \sum_i (x_i - \mu_X)^2 \cdot P(X = x_i)$$

Der Variationskoeffizient v ist bei einer Standardabweichung σ und einem Mittelwert $\bar{x}$ definiert als:

$$v = \frac{\sigma}{\bar{x}}$$

[33] Vgl. [YD16]

Außerdem wurden für die zu untersuchenden Lösungen zusätzlich die minimale und maximale Antwortzeit aufgeführt, um die absolute Streuung zu betrachten.

Dasselbe statistische Verfahren wurde auch beim Test der synthetischen Datenmenge verwendet, welche aus 100 Datensätzen besteht. Die Zusammensetzung der synthetischen Datenmenge verhält sich relativ zur realistischen Datenmenge und ist deswegen nach ihrem Komplexitätsgrad in sieben Stufen ansteigend gruppiert. In der Versuchsdurchführung werden die Datensätze jeweils 100 Mal gelesen. Pro Komplexitätsgruppe wurde der Erwartungswert (Mittelwert) und der Variationskoeffizient ermittelt, um Rückschlüsse auf das Leistungsverhalten bei zeitlich-räumlicher Skalierung der getesteten Implementierungen zu erhalten.[34]

5.2.3 Vergleich mittels Benchmarks

Die Fallstudie von Floratou et al. [Flo+12] unterscheidet sich in der Versuchsanordnung deutlich von den bisher vorgestellten Beispielen. Die Autoren untersuchten das Skalierungsverhalten von relationalen und NoSQL-Datenbanksystemen (in dieser Studie: Spaltenfamilien- und dokumentorientiert) beispielhaft anhand von zwei Benchmarks (TPC-H und YSCB), um damit ein möglichst großes Anwendungsspektrum aus dem Bereich Big Data abzubilden. Die Benchmarks sind so gewählt, dass die Arbeitslast möglichst realitätsnah ist. Für den Test von RDBMS wurde das TPC-H Benchmark ausgewählt, welches ein Entscheidungsunterstützungssystem (DSS) simuliert.[35] Beim YCSB-Framework, welches zum Test der nicht-relationalen Systeme verwendet wurde, sind zur Simulation einer datenintensiven Anwendung fünf Arbeitslasten definiert worden und sollen so OLAP- und OLTP-Anwendungen simulieren.[36]

Der Leistungsvergleich mittels TPC-H Benchmark erfolgte über die relationale Datenbank und die Spaltenfamilien-Datenbank. Sie bietet eine SQL-Schnittstelle an und ist somit auch für dieses Benchmark qualifiziert. Beim Test wurden die 22 lesenden Abfragen des TPC-H Benchmark mit unterschiedlichen Datenmengen (250 GB, 1 TB, 4 TB, 16 TB) jeweils zwei Mal ausgeführt. Diese Datenmengen werden in der Studie als TPC-H Skalierungsfaktoren bezeichnet. Aus der Summe der ermittelten Antwortzeiten pro Skalierungsfaktor berechnen die Autoren sowohl das standardisierte arithmetische als auch das standardisierte geometrische Mittel, welches bei einer zu erwartenden schiefen Verteilung der Daten einen aussagekräftigeren Mittelwert darstellen soll. Der geometrische Mittelwert $\bar{x}_{\text{geom}}$ ist definiert als:

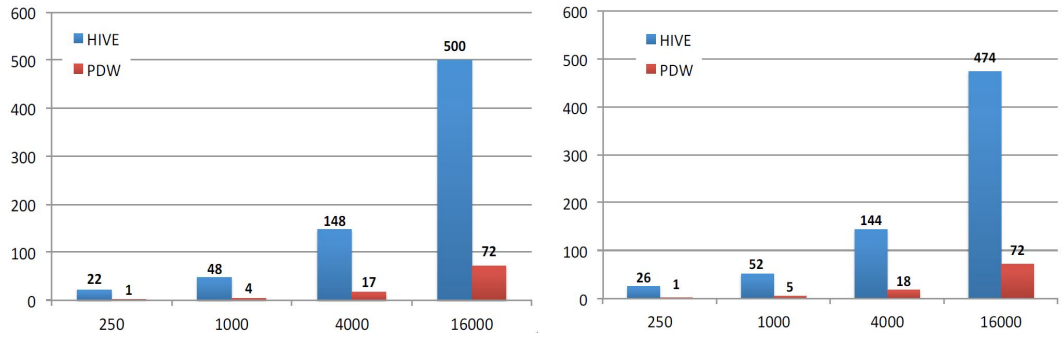
$$\bar{x}_{\text{geom}} = \sqrt[n]{\prod_{i=1}^{n} x_i}$$

Die Standardisierung der Mittelwerte ist notwendig, weil der Erwartungswert hinsichtlich der unterschiedlich großen Datenmengen und des gewählten Datenbanktyps variiert und dadurch eine Vergleichbarkeit gewährleistet werden soll. Die standardisierte Zufallsvariable Z mit einem Erwartungswert $E(X) = \mu$ und Varianz $V(X) = \sigma^2$ ist definiert als:

$$Z = \frac{X - \mu}{\sigma}$$

Die Abbildung 5.5(a)[37] zeigt das standardisierte arithmetische Mittel der Antwortzeiten für die unterschiedlichen TPC-H Skalierungsfaktoren beim Vergleich einer nicht-relationalen Lösung („HIVE“) und relationalen Lösung („PDW“). Die Ergebnisse sind auf

[34] Vgl. [Fio+16]
[35] Siehe hierzu auch Abschnitt 3.2.3
[36] Siehe hierzu auch Abschnitt 2.2
[37] Abbildung entnommen aus [Flo+12]



(a) Standardisiertes arithmetisches Mittel (b) Standardisiertes geometrisches Mittel

Abbildung 5.5: Ergebnisse für zwei Datenbanken im TPC-H Benchmark

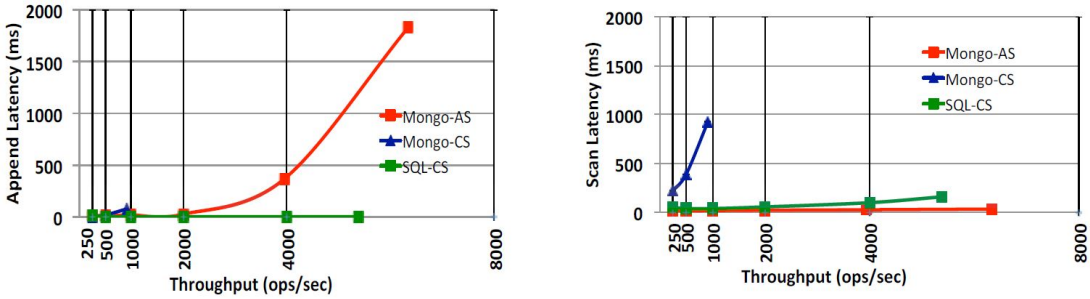


Abbildung 5.6: Arbeitslast (95% Scan-Operationen und 5% Append-Operationen)

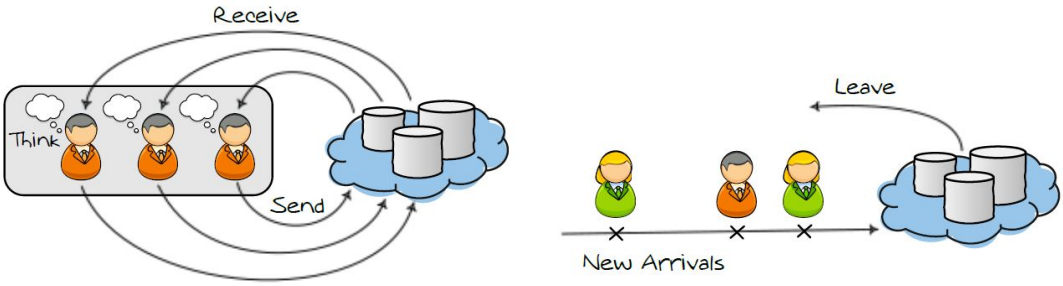
den Wert von PDW beim Skalierungsfaktor 250 normalisiert. Die Abbildung 5.5(b)[38] zeigt den gleichen Sachverhalt, jedoch wurde in diesem Fall das standardisierte geometrische Mittel für die Antwortzeiten verwendet.

Mit dem YCSB-Framework wurden drei Lösungen verglichen: eine relationale Datenbank sowie zwei Konfigurationen einer dokumentorientierten Datenbank. Alle drei Lösungen werden typischerweise für OLAP-/OLTP-Anwendungen eingesetzt. Zur Simulation einer solchen Anwendung wurden fünf Arbeitslasten definiert, die sich bezüglich ihrer prozentualen Zusammensetzung an Lese-, Schreib- und Scanoperationen unterscheiden. Darüber hinaus ist bei diesem Versuch erwähnenswert, dass durch den Test von zwei dokumentorientierten Datenbanken der Grad des Einflusses ihrer Konfigurationseinstellungen ermittelt wurde. In diesem Szenario wurde die Latenz von verschiedenen Datenoperationen unter steigender Anzahl auszuführender Datenoperationen pro Zeiteinheit verglichen. Im Ergebnis stellen die Autoren fest, dass relationale Datenbanken im standardisierten Test performanter als NoSQL-Datenbanken sind. Letztere besitzen jedoch die besseren Fähigkeiten zur Skalierung, wie in Abbildung 5.6[39] am Beispiel einer Arbeitslast zu sehen ist.[40]

5.3 Kritische Auseinandersetzung

Die Autoren des YCSB-Framework führen an, dass die Entwicklung eines Benchmarks zur Erhebung vergleichbarer Metriken nicht trivial ist. Dies geschieht vor dem Hintergrund, dass bisherige Leistungsvergleiche keine realistische Arbeitslast abbilden konnten und die

[38] Abbildung entnommen aus [Flo+12]
[39] Abbildung entnommen aus [Flo+12]
[40] Vgl. [Flo+12]



(a) Geschlossenes System (b) Offenes System

Abbildung 5.7: Geschlossenes und offenes System im Vergleich

Ergebnisse solcher Mikro- oder Makro-Benchmarks als nicht hinreichend aussagekräftig angesehen wurden.[41] Das YCSB-Framework ist, wie andere standardisierte Leistungsvergleiche (TPC) auch, als *geschlossenes System* konzipiert. Im Zusammenhang mit dem Test von Datenbanken ist damit gemeint, dass eine bestimmte Anzahl an Benutzern (Threads) wiederholend Abfragen gegen das zu testende Systeme absetzt, ein Ergebnis zu dieser Abfrage erhält und dieses weiterverarbeitet. Erst danach wird eine neue Abfrage durch einen Benutzer gestellt (siehe Abbildung 5.7(a)[42]). Im Gegensatz dazu steht ein *offenes System*, bei dem die Anfragen nicht in Intervallen, sondern in zufälliger zeitlicher Reihenfolge in einem bestimmten Zeitintervall gestellt werden, wie in Abbildung 5.7(b)[43] dargestellt. Ein offenes System entspricht auch dem beobachteten Anwendungsszenario, wie beispielsweise im technischen Betrieb von Sozialen Medien.[44]

5.3.1 Coordinated Omission Problem

Das *Coordinated Omission Problem* beschreibt den Fall, wenn in einem geschlossenen System die Abfragen synchron an das zu testende System abgesetzt werden und es gleichzeitig in dem zu testenden Datenbanksystem zu Unterbrechungen und Verzögerung bei der Anfragenverarbeitung kommt. Aufgrund der Synchronizität der Abfragen in einem geschlossenen System wird ein sogenannter „Hiccup“ (Schluckauf) nur unzureichend erfasst und verzerrten so die Ergebnisse der Leistungsmessung.[45]

Die Abbildung 5.8[46] verdeutlicht diese Verzerrung in der Leistungsmessung aufgrund fehlender Messwerte durch die Synchronizität der Abfragen. Hierfür wurde in der Fallstudie von Friedrich et al. [FWR17] ein Szenario geschaffen, welches einen Hiccup simuliert. Am Beispiel einer alten Version des YSCB-Frameworks (Messwerte als roter Kreis in der Abbildung dargestellt) wurde dieses Phänomen analysiert, in der die Abfragen synchron abgefeuert werden, und eines Werkzeuges (Messwerte dargestellt als blaues Kreuz), welches die Abfragen asynchron sendet.[47] Die Simulation eines Hiccups wurde durch Erweiterung des YCSB-Frameworks („SickStore“) bewerkstelligt, welches alle 30 Sekunden einen Hiccup von 1 Sekunde Länge nachahmt. Ebenfalls als Erweiterung des

[41] Vgl. [Coo+10]
[42] Abbildung entnommen aus [FWR17]
[43] Abbildung entnommen aus [FWR17]
[44] Vgl. [FWR17]
[45] Vgl. [FWR17]
[46] Abbildung entnommen aus [FWR17]
[47] Nach Aussage von Friedrich et al. ist in der aktuellen Version des YCSB-Frameworks die Implementierung des Testablaufs so abgeändert worden, dass das Coordinated Omission Problem gemindert wurde. Das grundsätzliche Problem ist nach Auffassung der Autoren damit nicht behoben, so dass das YSCB-Framework weiterhin als geschlossenes System betrachtet werden muss.

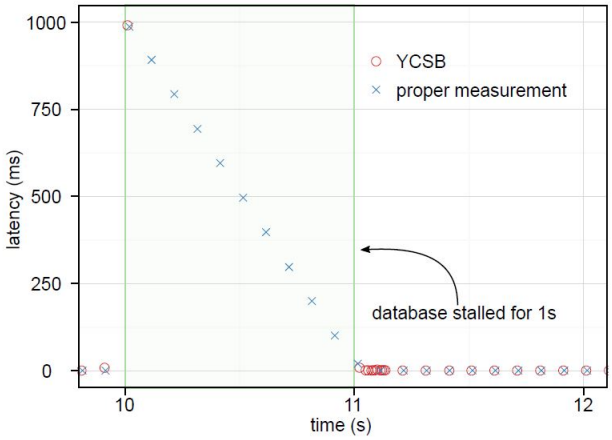


Abbildung 5.8: Das Coordinated Omission Problem

YCSB-Frameworks wurde ein Werkzeug („NoSQLMark“) erschaffen, welches asynchron die Abfragen verschickt und die Leistungsmessung zusätzlich durchführt.

5.3.2 Validierung des YCSB-Frameworks

Eine Leistungsanalyse von Datenbanken unter Zuhilfenahme von Werkzeugen wie dem YCSB-Framework basiert auf unterschiedlichen Annahmen bezüglich des zu testenden Datenbanksystems, der zugrunde liegenden Infrastruktur und Netzwerktopologie sowie des Testszenarios. Hierzu zeigen Wingerath et al. in einer kritischen Analyse des Benchmarkings von NoSQL-Datenbanken [Win+15], dass die erwähnten Annahmen, wenn sie nicht zutreffen, ungenaue Ergebnisse in der Leistungsmessung zur Folge haben. Darüber hinaus sei es wichtig, nicht nur das Messverfahren zu betrachten, sondern auch die Implementierung der Werkzeuge zur Leistungsanalyse kritisch zu hinterfragen. Beispielhaft hierfür steht der Umgang mit Transaktionen oder das Lesen veralteter Daten.[48]

Mit den Begriffen *Staleness* oder *Stale Read* bezeichnet man den Umstand, wenn bei einer Leseoperation bereits veraltet Datenobjekte oder Werte zurückgeliefert werden. Das geschieht immer dann, wenn in einem Datenbanksystem, welches aus mehreren replizierenden Knoten besteht, ein Datensatz nicht auf allen Knoten aktualisiert ist. Dieses Zeitfenster, in dem noch nicht alle Knoten aktualisiert sind, wird als *Staleness Window* bezeichnet. Diese Betrachtungsweise entspricht der Definition einer *Data-centric Staleness* und steht im Gegensatz zu einer *Client-centric Staleness*, wie sie im YCSB-Framework definiert ist[49]. In diesem Fall wird betrachtet, wie lange nach einer erfolgten Schreiboperation ein veralteter Datensatz zurückgegeben wird.[50]

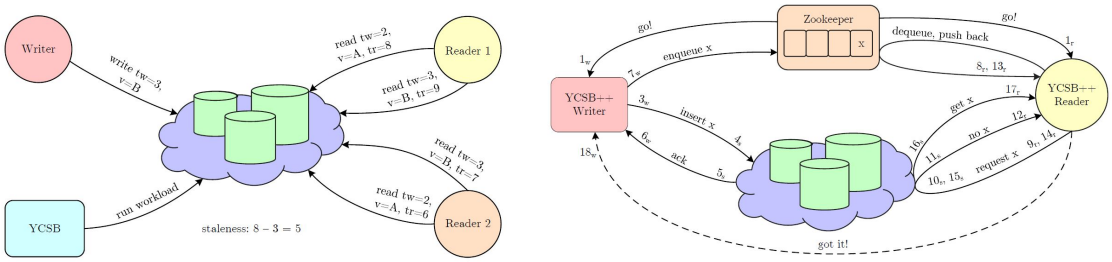
Es existieren für beide Betrachtungsweisen von Staleness unterschiedliche Verfahren zur Untersuchung mittels Einsatzes des YCSB-Frameworks. Beide Ansätze haben gemeinsam, dass sie keine exakten Werte liefern, sondern immer nur eine obere und untere zeitliche Grenze definieren, in welcher ein Lesen veralteter Daten möglich ist.

- **Data-centric Staleness:** Das Verfahren zur Betrachtung von Data-centric Staleness beruht auf der Annahme, dass in einem System bestehend aus verschiedenen Knoten und der YCSB-Instanzen für die lesenden und schreibenden Clients die Systemuhrzeiten vollkommen synchronisiert sind. Jeder Datensatz, der geschrieben wird,

[48] Vgl. [Win+15]

[49] Wingerath et al. beziehen sich hierbei auf die Standardversion des YCSB-Frameworks bestehend aus Tier 1 bis 4, wie in Abschnitt 3.2.1 beschrieben.

[50] Vgl. [Win+15]



(a) Aufbau des Messverfahrens für Data-centric Staleness

(b) Aufbau des Messverfahrens für Client-centric Staleness

Abbildung 5.9: Messverfahren zur Ermittlung der Staleness nach Schreiboperationen

ist mit einem Zeitstempel versehen. In diesem Modell errechnet sich das Zeitfenster für Staleness als Differenz zwischen dem letzten Stale Read und dem Zeitpunkt, an dem der Datensatz auf dem schreibenden Client generiert und abgesendet wurde, wie schematisch in Abbildung 5.9(a)[51] dargestellt. Trotz der vollkommen synchronisierten Systemuhrzeiten auf den Knoten und Clients liefert dieses Vorgehen nur einen angenäherten Wert, weil der Zeitstempel jener Zeitpunkt ist, bei dem der Datensatz im Client generiert wurde, bevor er übertragen und durch die Datenbank verarbeitet wurde. Jedoch ist für ein präzises Ergebnis der Zeitpunkt erforderlich, an dem der Datensatz in der Datenbank fertig geschrieben und verarbeitet wurde.[52]

- **Client-centric Staleness:** Das Verfahren zur Betrachtung von Client-centric Staleness basiert auf dem Prinzip der Benachrichtigung zwischen schreibendem und lesendem Client. In diesem Fall wird ein Dienst (Apache ZooKeeper[53]) genutzt, mit dem die Architektur einer Warteschlange implementiert wurde. Über Nachrichten wird der geschriebene Wert in zeitlichen Intervallen auf allen Knoten abgefragt. Findet er keine Übereinstimmung, gibt der lesende Client die Nachricht zurück in die Warteschleife, so dass diese zu einem späteren Zeitpunkt wieder als Nachricht an den lesenden Client gesendet wird. Findet der lesende Client eine Übereinstimmung, sendet er eine Nachricht an den schreibenden Client zur Bestätigung. In Abbildung 5.9(b)[54] ist zu sehen, dass die gewählte Architektur zur Leistungsmessung nachteilig ist, weil die Nachrichten durch ein Netzwerk gesendet werden und sich die Netzwerklatenz aufsummieren kann. Zusätzlich hierzu addiert sich je nach Last die Verweildauer in der Warteschlange des Dienstes.[55]

[51] Abbildung entnommen aus [Win+15]
[52] Vgl. [Win+15]
[53] `https://zookeeper.apache.org/`
[54] Abbildung entnommen aus [Win+15]
[55] Vgl. [Pat+11], [Win+15]

6 Fazit

Die vorliegende Thesis vermittelt ein strukturelles Verständnis des Leistungsbegriffs im Kontext von NoSQL-Datenbanken. Die Heranführung an den Begriff der Leistung erfolgte über die Anforderungen, die an nicht-relationale Datenbanksysteme gestellt werden, die verschiedenen Typen von NoSQL-Datenbanken und ihre typischen Eigenschaften. Ein Schwerpunkt lag in dieser Arbeit in der Vorstellung von speziellen Systemqualitäten für nicht-relationale Datenbanken und in der Beschreibung ihrer Konfigurationsmöglichkeiten. Am Beispiel der Schreib- und Lesegeschwindigkeit wurden konkret die Einflussmöglichkeiten verschiedener Konfigurationen aufgezeigt, wie zum Beispiel die Konfiguration von genutzten Speichermedien und das Loggingkonzept, welches für den persistenten Speicher verwendet wird.

Die Auswahl der vorgestellten Methoden und Vorgehensmodelle erfolgte nach dem Gesichtspunkt der wissenschaftlichen Relevanz und Akzeptanz. Gleichzeitig sollte die Anwendbarkeit und Zweckmäßigkeit für einen Leistungsvergleich von NoSQL-Datenbanken mittels Messung und Bewertung gegeben sein. Bei der Einführung in das technische Werkzeug, das YCSB-Framework, ist darauf geachtet worden, einen voll umfänglichen Überblick der Architektur und ihrer Erweiterungen zu geben. Diese Thesis versteht sich somit als Grundlage zu einem experimentellen Vorgehen, um Lösungen für Datenbanksysteme hinsichtlich ihres Leistungsverhaltens iterativ zu erforschen.

Die Betrachtung der Leistungsmessung im Querschnitt widmete sich insbesondere der Beschreibung der Arbeitslast aufgrund ihrer Relevanz im Testaufbau: Sie sollte das Abbild des realistischen Anwendungsszenarios sein. Der Aspekt der Schreib- und Lesegeschwindigkeit, welcher sich durch alle Kapitel zieht, wurde insgesamt ausführlich beschrieben, wenngleich eine inhaltliche Abgrenzung in den jeweiligen Kapitel schwierig war. Die Leistungsanalyse von NoSQL-Datenbanken ein sehr junges Forschungsfeld ist und erst mit Erscheinen des YCSB-Frameworks Gegenstand wissenschaftlicher Untersuchung geworden ist. Weitere wiederkehrende Bezugspunkte innerhalb dieser Arbeit sind die Skalierbarkeit und Datenkonsistenz. Diese Aspekte werden ausführlich im Kapitel Leistungsbewertung der Rohdaten und Metriken behandelt und haben dadurch einen konkreten Bezug zu anwendungsbezogenen Fragestellungen. Die vorgestellten Verfahren versuchen den aktuell verfügbaren Stand der Literatur wiederzugeben (Stand: Mai 2017).

An dieser Arbeit ist kritisch zu betrachten, dass der Leistungsbegriff für NoSQL-Datenbanken ausschließlich über quantitative Merkmale, wie Datenmenge und Ausführungsgeschwindigkeit, definiert wurde. Dies liegt zum einen an der inhaltlichen Abgrenzung dieser Arbeit und zum anderen auch an der Auswahl der zur Verfügung stehenden Literatur. Die systematische Untersuchung qualitativer Merkmale eines Datenbanksystems, wie zum Beispiel die Mächtigkeit der zur Verfügung stehenden Abfragesprachen und Schnittstellen für eine Datenbankimplementierung, wird hoffentlich in Zukunft verstärkt wissenschaftliche Beachtung finden wird. Weil Qualitätsdefinitionen im Vordergrund standen, soll diese Literaturstudie als Ermutigung verstanden werden, die theoretische Ausführung mit konkreten Fallstudien zu untermauern. Die Validierung dieses Methoden- und Werkzeugkastens kann Gegenstand weiterer wissenschaftlicher Untersuchungen sein.

Für die Zukunft ist anzunehmen, dass die Popularität von NoSQL-Datenbanksystemen

weiter ansteigen und dies zu einer weiteren technologischen Evolution in diesem Bereich führen wird. Interessant wird hierbei sein, in denen nicht-relationale Datenbanken in Bereiche vorrücken können, wo bisher relationale Datenbanken beim Leistungsvergleich besser abschneiden und ob umgekehrt, ob relationale Datenbanken in jene Bereiche aufschließen können, in denen NoSQL-Datenbanken derzeit leistungsmäßig führend sind.

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ACID	Atomicity, Consistency, Isolation und Durability
API	Application Programming Interface
BASE	Basically Available, Soft State, Eventually Consistent
CAD	Computer Aided Design
CAP	Consistency, Availability, Partition Tolerance
CEEM	Cloud Evaluation Experiment Methodology
CPU	Central Processing Unit
CRUD	Create, Read, Update, Delete
DAO	Data Access Object
DB	Datenbank
DBMS	Datenbankmanagementsystem
DOE	Design Of Experiments
DSS	Decision Support System
GIS	Geoinformationssystem
HDD	Hard Disk Drive
HTTP	Hypertext Transfer Protocol
IT	Informationstechnologie
JNI	Java Native Interface
JSON	JavaScript Object Notation
LEAP4BD	Lightweight Evaluation and Prototyping for Big Data
NoREL	Nicht-relationales Datenbanksystem
NoSQL	Not Only SQL
OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
ORM	Objekt-relationales Mapping
RAM	Read-Access Memory
RDBMS	Relationales Datenbankmanagementsystem
REST	Representational State Transfer
SQL	Structured Query Language
SSD	Solid-State Drive
SUT	System Under Test
TPC	Transaction Processing Performance Council
UML	Unified Modeling Language
WORM	Write-Once-Read-Many
XML	Extensible Markup Language
YCSB	Yahoo! Cloud Serving Benchmark

Literaturverzeichnis

[ABF14a] Veronika Abramova, Jorge Bernardino und Pedro Furtado. „Testing Cloud Benchmark Scalability with Cassandra“. In: *2014 IEEE World Congress on Services, SERVICES 2014, Anchorage, AK, USA, June 27 - July 2, 2014*. 2014, S. 434–441. URL: http://dx.doi.org/10.1109/SERVICES.2014.81.

[ABF14b] Veronika Abramova, Jorge Bernardino und Pedro Furtado. „Which NoSQL Database? A Performance Overview“. In: *Open Journal of Databases (OJDB)* 1.2 (2014), S. 17–24. ISSN: 2199-3459. URL: https://www.ronpub.com/OJDB-v1i2n02_Abramova.pdf.

[Ame+16] Parinaz Ameri, Nico Schlitter, Jörg Meyer und Achim Streit. „NoWog: A Workload Generator for Database Performance Benchmarking.“ In: *DASC / PiCom / DataCom / CyberSciTech*. IEEE Computer Society, 2016, S. 666–673. ISBN: 978-1-5090-4065-0. URL: http://dblp.uni-trier.de/db/conf/dasc/dasc2016.html#AmeriSMS16.

[BBF14] Melyssa Barata, Jorge Bernardino und Pedro Furtado. „YCSB and TPC-H: Big Data and Decision Support Benchmarks“. In: *2014 IEEE International Congress on Big Data, Anchorage, AK, USA, June 27 - July 2, 2014*. 2014, S. 800–801.

[Bon00] André B. Bondi. „Characteristics of Scalability and Their Impact on Performance“. In: *Proceedings of the 2Nd International Workshop on Software and Performance*. WOSP '00. New York, NY, USA: ACM, 2000, S. 195–203. ISBN: 1-58113-195-X. URL: http://doi.acm.org/10.1145/350391.350432.

[Bou+10] Stefan Bouckaert, Jono Vanhie-Van Gerwen, Ingrid Moerman, Stephen C Phillips, Jerker Wilander, Shafqat Ur Rehman, Walid Dabbous und Thierry Turletti. „Benchmarking computers and computer networks“. In: *EU FIRE White Paper* (2010).

[Coo+10] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan und Russell Sears. „Benchmarking cloud serving systems with YCSB“. In: *Proceedings of the 1st ACM symposium on Cloud computing*. SoCC '10. New York, NY, USA: ACM, 2010, S. 143–154. ISBN: 978-1-4503-0036-0. URL: http://doi.acm.org/10.1145/1807128.1807152.

[Cooa] Brian F. Cooper. *Yahoo! Cloud Serving Benchmark – Overview and results*. http://www.brianfrankcooper.net/home/publications/ycsb-v4.pdf. Besucht am: 02.04.2017.

[Coob] Brian Frank Cooper. *Yahoo! Cloud Serving Benchmark (YCSB)*. https://github.com/brianfrankcooper/YCSB. Besucht am: 31.03.2017.

[Coua] Transaction Processing Performance Council. *TPC Benchmark C – Standard Specification Revision 5.11*. http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-c_v5.11.0.pdf. Besucht am: 06.04.2017.

[Coub] Transaction Processing Performance Council. *TPC Benchmark E – Standard Specification Version 1.14.0*. http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-e_v1.14.0.pdf. Besucht am: 06.04.2017.

[Couc] Transaction Processing Performance Council. *TPC Benchmark H – Standard Specification Revision 2.17.1*. http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.17.1.pdf. Besucht am: 02.04.2017.

[DDS16] Nitin Dangwal, Neha Mehrotra Dewan und Sonal Sachdeva. „Testing the Cloud and Testing as a Service“. In: *Encyclopedia of Cloud Computing*. John Wiley & Sons, 2016, S. 338–348. ISBN: 978-1-118-82197-8.

[Dey+14] Akon Dey, Alan Fekete, Raghunath Nambiar und Uwe Röhm. „YCSB+T: Benchmarking web-scale transactional databases“. In: *Data Engineering Workshops (ICDEW), 2014 IEEE 30th International Conference on*. 2014, S. 223–230. URL: http://dx.doi.org/10.1109/ICDEW.2014.6818330.

[DFR15] Akon Dey, Alan Fekete und Uwe Röhm. „Scalable distributed transactions across heterogeneous stores“. In: *2015 IEEE 31st International Conference on Data Engineering*. 2015, S. 125–136. URL: http://dx.doi.org/10.1109/ICDE.2015.7113278.

[EN10] Ramez Elmasri und Shamkant Navathe. *Fundamentals of Database Systems*. 6. Aufl. Addison-Wesley Publishing Company, 2010. ISBN: 978-0-136-08620-8.

[Fio+16] Sara Fioravanti, Simone Mattolini, Fulvio Patara und Enrico Vicario. „Experimental Performance Evaluation of different Data Models for a Reflection Software Architecture over NoSQL Persistence Layers.“ In: *ICPE*. ACM, 2016, S. 297–308. ISBN: 978-1-4503-4080-9. URL: http://dblp.uni-trier.de/db/conf/wosp/icpe2016.html#FioravantiMPV16.

[Flo+12] Avrilia Floratou, Nikhil Teletia, David J. DeWitt, Jignesh M. Patel und Donghui Zhang. „Can the Elephants Handle the NoSQL Onslaught?“ In: *Proc. VLDB Endow.* 5.12 (Aug. 2012), S. 1712–1723. ISSN: 2150-8097. URL: http://dx.doi.org/10.14778/2367502.2367511.

[FWR17] Steffen Friedrich, Wolfram Wingerath und Norbert Ritter. „Coordinated Omission in NoSQL Database Benchmarking“. In: *Datenbanksysteme für Business, Technologie und Web (BTW 2017), 17. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme"(DBIS), 6.-10. März 2017, Stuttgart, Germany, Workshopband*. 2017, S. 215–225.

[Ges+16] Felix Gessert, Wolfram Wingerath, Steffen Friedrich und Norbert Ritter. „NoSQL Database Systems: A Survey and Decision Guidance“. In: *Computer Science - Research and Development* (2016).

[GL02] Seth Gilbert und Nancy Lynch. „Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-tolerant Web Services“. In: *SIGACT News* 33.2 (Juni 2002), S. 51–59. ISSN: 0163-5700. URL: http://doi.acm.org/10.1145/564585.564601.

[Iye16] Ganesh Neelakanta Iyer. „Cloud Testing: An Overview“. In: *Encyclopedia of Cloud Computing*. John Wiley & Sons, 2016, S. 327–337. ISBN: 978-1-118-82197-8.

[Kle+15] John Klein, Ian Gorton, Neil Ernst, Patrick Donohoe, Kim Pham und Chrisjan Matser. „Performance Evaluation of NoSQL Databases: A Case Study“. In: *Proceedings of the 1st Workshop on Performance Analysis of Big Data Systems*. PABS '15. Austin, Texas, USA: ACM, 2015, S. 5–10. ISBN: 978-1-4503-3338-2. URL: http://doi.acm.org/10.1145/2694730.2694731.

[Koo] James Koopmann. *What Is Your Definition of Database Workload?* http://www.databasejournal.com/features/oracle/article.php/3794731/What-Is-Your-Definition-of-Database-Workload.htm. Besucht am: 20.03.2017.

[LM13] Yishan Li und Sathiamoorthy Manoharan. „A performance comparison of SQL and NoSQL databases“. In: *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*. 2013, S. 15–19.

[LOR16] Zheng Li, Liam O'Brien und Rajiv Ranjan. „Cloud Service Evaluation“. In: *Encyclopedia of Cloud Computing*. John Wiley & Sons, 2016, S. 349–360. ISBN: 978-1-118-82197-8.

[LOZ13] Zheng Li, Liam O'Brien und He Zhang. „CEEM: A Practical Methodology for Cloud Services Evaluation“. In: *2013 IEEE Ninth World Congress on Services*. 2013, S. 44–51.

[MG11] Peter Mell und Timothy Grance. *Recommendations of the National Institute of Standards and Technology – The NIST Definition of Cloud Computing*. 2011. URL: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf (besucht am 05. 12. 2016).

[MK16] Andreas Meier und Michael Kaufmann. *SQL- & NoSQL-Datenbanken*. 8. Aufl. eXamen.press. Springer Berlin Heidelberg, Springer Vieweg, 2016. ISBN: 978-3-662-47664-2. URL: http://dx.doi.org/10.1007/978-3-662-47664-2.

[OVC16] Fábio Roberto Oliveira und Luis M. del Val Cura. „Performance Evaluation of NoSQL Multi-Model Data Stores in Polyglot Persistence Applications“. In: *Proceedings of the 20th International Database Engineering & Applications Symposium, IDEAS 2016, Montreal, QC, Canada, July 11-13, 2016*. 2016, S. 230–235. URL: http://doi.acm.org/10.1145/2938503.2938518.

[Pat+11] Swapnil Patil, Milo Polte, Kai Ren, Wittawat Tantisiriroj, Lin Xiao, Julio López, Garth Gibson, Adam Fuchs und Billie Rinaldi. „YCSB++: Benchmarking and Performance Debugging Advanced Features in Scalable Table Stores“. In: *Proceedings of the 2Nd ACM Symposium on Cloud Computing*. SOCC '11. Cascais, Portugal: ACM, 2011, 9:1–9:14. ISBN: 978-1-4503-0976-9. URL: http://doi.acm.org/10.1145/2038916.2038925.

[Sak16] Sherif Sakr. „Cloud-Hosted Databases“. In: *Encyclopedia of Cloud Computing*. John Wiley & Sons, 2016, S. 562–571. ISBN: 978-1-118-82197-8.

[Sch12] Jiri Schindler. „I/O Characteristics of NoSQL Databases“. In: *PVLDB* 5.12 (2012), S. 2020–2021. URL: http://vldb.org/pvldb/vol5/p2020_jirischindler_vldb2012.pdf.

[Sch13] Jiri Schindler. „Profiling and analyzing the I/O performance of NoSQL DBs“. In: *ACM SIGMETRICS / International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS '13, Pittsburgh, PA, USA, June 17-21, 2013*. 2013, S. 389–390. URL: http://doi.acm.org/10.1145/2465529.2479782.

[SE16] Surya Narayanan Swaminathan und Ramez Elmasri. „Quantitative Analysis of Scalable NoSQL Databases“. In: *2016 IEEE International Congress on Big Data, San Francisco, CA, USA, June 27 - July 2, 2016*. 2016, S. 323–326. URL: http://dx.doi.org/10.1109/BigDataCongress.2016.49.

[SGR15] Stephan Schmid, Eszter Gálicz und Wolfgang Reinhardt. „Performance investigation of selected SQL and NoSQL databases“. In: *Proceedings 2015. The 18th AGILE International Conference on Geographic Information Science, Lisbon, 9-12 June 2015. Geographic Information Science as an Enabler of Smarter Cities and Communities.* 2015.

[SKS11] Abraham Silberschatz, Henry F. Korth und S. Sudarshan. *Database System Concepts*. McGraw-Hill, 2011. ISBN: 978-0-071-28959-7. URL: https://books.google.de/books?id=Hvn_QQAACAAJ.

[Tiw11] Shashank Tiwari. *Professional NoSQL*. Wiley, 2011. ISBN: 978-0-470-94224-6. URL: https://books.google.it/books?id=tv5iO9MnObUC.

[Wie15] Lena Wiese. *Advanced data management: for SQL, NoSQL, Cloud and distributed databases*. De Gruyter graduate. De Gruyter Oldenbourg, 2015. ISBN: 978-3-11-044140-6. URL: http://digitool.hbz-nrw.de:1801/webclient/DeliveryManager?pid=6543746.

[Win+15] Wolfram Wingerath, Steffen Friedrich, Felix Gessert und Norbert Ritter. „Who Watches the Watchmen? On the Lack of Validation in NoSQL Benchmarking“. In: *Datenbanksysteme für Business, Technologie und Web (BTW), 16. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme"(DBIS), 4.-6. März 2015 in Hamburg, Germany. Proceedings*. 2015, S. 351–360. URL: http://subs.emis.de/LNI/Proceedings/Proceedings241/article20.html.

[YD16] Amal W. Yassien und Amr F. Desouky. „RDBMS, NoSQL, Hadoop: A Performance-Based Empirical Analysis“. In: *Proceedings of the 2Nd Africa and Middle East Conference on Software Engineering*. Cairo, Egypt: ACM, 2016, S. 52–59. ISBN: 978-1-4503-4293-3. URL: http://doi.acm.org/10.1145/2944165.2944174.

Erklärung

Hiermit versichere ich, *Tiziano Lombardi*, dass ich die vorliegende Thesis *„Leistungsmessung und Leistungsbewertung von NoSQL-Datenbanken“* ohne fremde Hilfe angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Stellen, die wörtlich oder sinngemäß einer Veröffentlichung entnommen wurden, sind als solche kenntlich gemacht. Die Arbeit hat in dieser oder ähnlicher Form im Rahmen einer anderen Prüfung noch nicht vorgelegen.

Berlin, im Mai 2017

Tiziano Lombardi